INFORMATICA, 2021, Vol. 32, No. 3, 619–660
© 2021 *Vilnius University*
DOI: https://doi.org/10.15388/21-INFOR454

A Systematic Mapping Study on Analysis of Code Repositories

Jaime SAYAGO-HEREDIA[1,*], Ricardo PÉREZ-CASTILLO[2], Mario PIATTINI[2]

[1] *Pontificia Universidad Católica del Ecuador, Sede Esmeraldas, Espejo y subida a Santa Cruz Casilla 08-01-0065, Ecuador*
[2] *Information Technology & Systems Institute, University of Castilla-La Mancha, Paseo de la Universidad, 4, 13071, Ciudad Real, Spain*
e-mail: jaime.sayago@pucese.edu.ec, ricardo.pdelcastillo@uclm.es, mario.piattini@uclm.es

Received: October 2020; accepted: May 2021

Abstract. Code repositories contain valuable information, which can be extracted, processed and synthesized into valuable information. It enabled developers to improve maintenance, increase code quality and understand software evolution, among other insights. Certain research has been made during the last years in this field. This paper presents a systematic mapping study to find, evaluate and investigate the mechanisms, methods and techniques used for the analysis of information from code repositories that allow the understanding of the evolution of software. Through this mapping study, we have identified the main information used as input for the analysis of code repositories (commit data and source code), as well as the most common methods and techniques of analysis (empirical/experimental and automatic). We believe the conducted research is useful for developers working on software development projects and seeking to improve maintenance and understand the evolution of software through the use and analysis of code repositories.

Key words: code repository analysis, repository mining, code repository, GitHub, systematic mapping study.

1. Introduction

Software engineering researchers have sought to optimize software development by analysing software repositories, especially code repositories. Code repositories contain important information about software systems and projects to analyse and process (Hassan, 2008). Code repositories contain valuable information, which can be extracted, processed and synthesized into output or resultant information. Information allows developers to improve maintenance, increase code quality and understand software evolution. For some years now, software engineering researchers have been working on extracting this information to support the evolution of software systems, improve software design and reuse, and empirically validate new ideas and techniques (Amann *et al.*, 2015).

[*] Corresponding author.

Researchers have a real challenge with the realization of these studies, since it is complex to analyse the different artifacts contained in the code repositories. Despite the challenge of analysing code repositories, they can provide solutions to problems that arise in a software development project such as defects, effort estimation, cloning, evolutionary patterns. Understanding these issues, along with other parameters and metrics obtained from the repository, can decrease maintenance costs and increase the quality of the software.

The objective of this Systematic Mapping Study (SMS) is to find, evaluate and investigate the mechanisms, methods and techniques used for the analysis of information from code repositories that allow the understanding of the evolution of software and research of this area. The primary studies for our research were taken from the main digital databases. The process of searching, analysing, and debugging the literature on code repositories was carried out through rigorous protocols and methodologies described (Section 4) in subsequent sections of the study. We obtained 236 documents out of a total of 3755 documents published between 2012 and 2019. This selected period (seven years) is a reasonable time period to avoid the selection of outdated, general or extensive works (Cosentino *et al.*, 2017; Tahir *et al.*, 2013), but also to prevent studies as a result of fashion peaks in a very short period (De Farias *et al.*, 2016). The selected studies allowed us to learn about the conducted research in this field and to answer the six research questions we posed.

This study reveals some trends in the current use of software coding evolution and the massive use of code repositories as a platform for software development. We believe this research is useful for developers who are working in software development projects, seek to improve maintenance and understand the evolution of software through the use and analysis of code repositories. These repositories included the source code and information about the development process, which can be analysed and used for both developers and project managers.

An important contribution is that we have defined a taxonomy which was divided according to the input, method and output of the studies and which is a part of our research. Through this mapping study, we have identified the main information inputs used in the analysis of code repositories, as well as the use of a wide variety of tools and methods for processing the information extracted from the code repository. Specifically, most of the studies focus on the use of empirical and other experimental analyses used in other research fields such as artificial intelligence, although there are plenty of other analysis methods employed. The study allows us to understand how the analysis of code repositories has evolved over the last decade. The scientific community has been constantly investigating the potential benefits of code repository analysis for a decade to understand the evolution of software, along with the possibility of validating techniques and tools (Amann *et al.*, 2015). It allows us to identify areas where researchers need to go deeper and find new lines of future research.

The rest of this paper is structured as follows: Section 2 provides a brief background on the definition and evolution of code repositories. Section 3 details the research methodology. Section 4 then describes the systematic mapping method applied in this study. Section 5 presents the results of the systematic mapping. Section 6 discusses the main results of the study and analyses them. Finally, Section 6 presents the conclusions of this study.

2. Background

The following is a description of the state of the art code repositories, showing the most important concepts and evolution of this knowledge area. In addition, this section shows papers on Systematic Literature Reviews (SLR), mapping studies and surveys.

2.1. *Code Repository Analysis*

One important task in this discipline is software comprehension, since software must be sufficiently understood before it can be properly modified and evolved. Actually, some authors argue that around 60% of software engineering effort is about software comprehension (Cornelissen *et al.*, 2009). Researchers in this area use different methods, artifacts and tools to analyse the source code and extract relevant knowledge (Chen *et al.*, 2016; Chahal and Saini, 2016). The analysis and understanding of software are complicated, alongside the handling of the different versions of the software and other information of the software development projects. To mitigate such problem, there are systems for controlling those versions, servers, and code repositories, and other software artifacts in general.

- **Version Control Systems (VCS).** Version Control Systems (VCS) is a tool that organizes the source code of software systems. VCS are used to store and build all the different versions of the source code (Ball *et al.*, 1997). In general, a VCS manages the development of an evolving object (Zolkifli *et al.*, 2018), recording every change made by software developers. In the process of building software, developers make changes in portions of the source code, artifacts, and the structure of the software. Thus, it is difficult to organize and document this process because it becomes a large and complex software. Therefore, VCS is a tool that allows developers to manage and control the process of development, maintainability and evolution of a software (Costa and Murta, 2013).
- **Software repositories.** Systems that store project data, e.g. issue control systems and version control systems, are known as software repositories (Falessi and Reichel, 2015). Software repositories are virtual spaces where development teams generate collaborative artifacts from the activities of a development process (Arora and Garg, 2018; Güemes-Peña *et al.*, 2018; Ozbas-Caglayan and Dogru, 2013). Software repositories contain large amount of software historical data that can include valuable information on the source code, defects, and other issues like new features (De Farias *et al.*, 2016). Moreover, we can extract many types of data from repositories, study them, and can make changes according to the need (Siddiqui and Ahmad, 2018). Due to open source, the number of these repositories and its uses is increasing at a rapid rate in the last decade (Amann *et al.*, 2015; Costa and Murta, 2013; Wijesiriwardana and Wimalaratne, 2018). Such repositories are used to discover useful knowledge about the development, maintenance and evolution of software (Chaturvedi *et al.*, 2013; Farias *et al.*, 2015). It is important to identify software repositories. Hassan (2008) describes the various examples of software repositories such as the following: historical repositories, runtime repositories, code repositories. Our research mainly focuses on code repositories.

- **Code repositories**. Code repositories are maintained by collecting source code from a large number of heterogeneous projects (Siddiqui and Ahmad, 2018). Code repositories like SourceForge, GitHub, GitLab, Bitbucket and Google Code contain a lot of information (Güemes-Peña *et al.*, 2018). These companies offer services that go beyond simple hosting and version control of the software (Joy *et al.*, 2018). Therefore, source code repositories have been attracting a huge interest from many researchers (Lee *et al.*, 2013).

These kinds of systems have been adopted by the industry and are used by a significant number of open source projects (Joy *et al.*, 2018). Thereby, such systems have become an important source of technical and social information about software development that is used to identify conventions, patterns, artifacts, etc. made by software development teams to understand and improve the quality of software (Del Carpio, 2017). However, the repository platforms only allow searches on projects, so they do not allow any analysis or value-added information to support the decision-making process (Hidalgo Suarez *et al.*, 2018). Researchers are interested in analysing these code repositories for information on different software issues (e.g. quality, defects, effort estimation, cloning, evolutionary patterns, etc.). Analysing code repositories is a difficult task that requires certain knowledge on how to access, gather, aggregate and analyse the vast amount of data in code repositories (Dyer *et al.*, 2015). Our research focuses on performing an SMS to know what kind of research has been done on the analysis of code repositories and to know what research areas have not been covered yet.

2.2. *Related Work*

This section describes some secondary studies (e.g. SMS, SLR and surveys) about the analysis of code repositories. To the best of our knowledge, in the relevant literature there are few SLR or SMS studies that tackle analysis of code repositories. We can find some works whose aim is to provide the state of the art in the field of code repository analysis.

In this line, De Farias *et al.* (2016), Siddiqui and Ahmad (2018) and Costa and Murta (2013), present reviews to investigate the different approaches of Mining Software Repositories (MSR), showing they are used for many purposes, mainly for understanding the defects, analysing the contribution and behaviour of developers, and understanding the evolution of software. In addition, the authors strive to discover the problems encountered during the development of software projects and the role of mining software repositories in solving those problems. A comparative study of data mining tools and techniques for extracting software repositories is also presented, one of these tools being VCS. These results can help practitioners and researchers to better understand and overcome version control system problems, and to devise more effective solutions to improve version control in a distributed environment. Zolkifli *et al.* (2018) discusses the background and work related to VCS that has been studied by researchers. The purpose of this document is to convey the knowledge and ideas that have been established in VCS. It is also important to understand the approaches to VCS, as different approaches will affect the software development process differently. Kagdi *et al.* (2007) presents a study on approaches to MSRs that includes sources such as information stored in VCS, error tracking requirements/systems, and communication files. The study provides a taxonomy of software repositories in the context of the evolution of software, which supports the development of tools, methods and processes to understand the evolution of software. In addition, Demeyer *et al.* (2013) provides an analysis of the MSR conference. This paper reports on technologies that are obsolete or emerging and current research methods for the date (2013) the study was conducted. In conclusion, the research focuses on the change and evolution of software, along with a few studies for the industry. The study already mentions the code repositories and their importance as an important source of data for software analysis.

This work focuses on the concepts presented in Section 2.1. Consequently, the research efforts (Costa and Murta, 2013; De Farias *et al.*, 2016; Siddiqui and Ahmad, 2018; Zolkifli *et al.*, 2018) are at the coarse-grained level where a generalized taxonomy of the different types of information analysed in software repositories is performed along with their respective tools and techniques for information extraction. This allows to have a general vision of the different code repositories, but it does not provide details of the information that is obtained from these software repositories. For example, what is the resulting information used for? What problems does it solve? and other questions linked to the maintenance and evolution of the software.

Other studies such as Amann *et al.* (2015) and Güemes-Peña *et al.* (2018), show that the main objectives of software repositories are mainly productivity objectives, such as identifying the impacts of change, as well as making development more effective. Other objectives are to support quality assurance, for example, by finding and predicting errors, or by detecting code clones and calculating the testing effort. Management objectives, such as estimating change effort, understanding human factors, or understanding processes, are also pursued, but in far fewer studies. In addition, in their research on the use of software repositories, they have identified the most relevant problems in the software development community: software degradation, dependencies between exchanges, error prediction and developer interaction. They pointed out that repositories record large volumes of data, although standard data collection tools are not available to extract specific data from the repositories. Most of the data sets came from open source projects with few contributions from industry participants.

In general, those studies highlight the challenge for researchers of analysing code repositories, as they need to deal with various software engineering artifacts, data sources, consider the human factor as a primary component, understand the areas of research and identify their current objectives, gaps and deficiencies, as well as to understand how to better evaluate their purposes and results.

GitHub is the main tool for software repositories with 79 million repositories (Borges and Tulio Valente, 2018). Cosentino *et al.* (2017), Kalliamvakou *et al.* (2016) analysed it through a systematic mapping of software development with GitHub, in which most of the work was focused on the interaction around the tasks related to coding and project communities. Some concerns were also identified about the reliability of these results because, in general, the proposal used small data sets and poor sampling techniques, employed a narrow range of methodologies and/or was difficult to understand. They also documented the results of an empirical study aimed at understanding the characteristics of software repositories such as GitHub; their results indicate that while GitHub is a rich source of data on software development, the use of GitHub for research purposes should take into account several potential hazards. Some potential dangers are manifested in relation to repository activity; there should also be a call for quantitative studies to be complemented by qualitative data. There are gaps in the data that may jeopardize the conclusions of any rigorous study. This software repository is a unique resource and continues to grow at a rapid rate; its users are finding innovative ways to use it and it will continue to be an attractive source for research in software engineering.

Therefore, in this section we can observe that SLRs, SMS, survey of the literature and text mining obtain information from MSR conferences or focus directly on the analysis of software repositories, with the purpose of knowing the software development process and understanding the evolution of software (Table 1). However, those studies do not analyse in detail other subsets that are a part of software repositories, such as code repositories, which require a greater emphasis of studies in terms of information obtained, tools, techniques, methodologies or information derived from these analyses, which will be identified in this study.

Consequently, in order to understand and identify the information obtained, tools, techniques and utilization of the software repository and its different research topics that remain to be covered, we perform an SMS that provides us with a complete view through different perspectives and does not follow a systematic process of document selection and data extraction, but rather a complete analysis validating the different approaches and proposals of various researchers.

3. Research Methodology

Based on the problem identified in the previous section, we prepared the main research question as follows:

RQ. What are the state of the art techniques and methods for the analysis of information from code repositories?

SMS is a secondary study that aims to classify and thematically analyse previous research (Kitchenham, 2007; Petersen *et al.*, 2008). It is related to a broader secondary study, a systematic literature review (SLR), which aims to gather and evaluate all research results on a selected research topic (de Almeida Biolchini *et al.*, 2007; Kitchenham *et al.*, 2009). There are several SLR methodologies, e.g. PRISMA y PRISMA – P2015 (Preferred Reporting Items for systematic reviews and meta-analyses for protocols 2015) (Shamseer *et al.*, 2015) which can be considered as a superior option, however, there are weaknesses (Haddaway *et al.*, 2018).

SMS usually use more general search terms, and aim to classify and structure the research field, whereas the aim of SLR is to summarise and conclusively evaluate the research results. Kitchenham (2007) also discuss the applications and state that SMS may

Table 1
Summary of the related work.

Paper	Type study	Objective	Extracted Info	Purpose
De Farias *et al.* (2016), Siddiqui and Ahmad (2018), Costa and Murta (2013)	SMS	Understand the defects, analyse the contribution and behaviour of developers, and understand the evolution of software	Software repositories, MSR	Software evolution
Zolkifli *et al.* (2018), Kagdi *et al.* (2007), Demeyer *et al.* (2013)	Systematic literature review, survey of the literature, text mining	Understand approaches to VCS, taxonomy of software repositories, analysis of obsolete or emerging technologies and current research methods	MSR, VCS, Conference MSR	Software development process, understanding the evolution of software, the change and evolution of software
Amann *et al.* (2015), Güemes-Peña *et al.* (2018)	Systematic literature review,	identification of impact change, maintainability, software quality, developer effort and bug prediction	Conference MSR	Data mining, machine learning, software process
Borges and Tulio Valente (2018), Cosentino *et al.* (2017), Kalliamvakou *et al.* (2016)	SMS	Coding and project communities, characteristics of software repositories	GitHub	Analysis Software Repository

be particularly suitable if only a few literature reviews have been conducted on the selected topic, and an overview of the field is sought.

Regardless of the selection, both approaches can be used to identify research gaps in the current state of research, but SMS is usually more applicable if the problem or topic is more generic (Kasurinen and Knutas, 2018). In addition, SMS can analyse what kind of studies have been conducted in the field, and what are their methods and results (Bailey *et al.*, 2007). In Fig. 1 we present the systematic mapping process proposed by Petersen *et al.* (2008) for the field of software engineering.

The goal of our SMS is to discover and evaluate the methods and techniques used for the analysis of code repository information that allow understanding the evolution of this research area of software engineering.

We have performed the SMS following the formal procedures defined by Petersen *et al.* (2015, 2008) and Kitchenham *et al.* (2011) and several steps of the standard process for SMS are presented in Section 3.1, while Section 3.2 describes the execution phase.

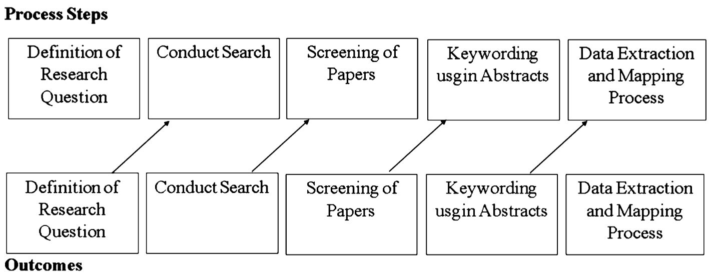
Process Steps
Definition of Research Question
Conduct Search
Screening of Papers
Keywording usgin Abstracts
Data Extraction and Mapping Process
Definition of Research Question
Conduct Search
Screening of Papers
Keywording usgin Abstracts
Data Extraction and Mapping Process
Outcomes

Fig. 1. Results obtained from the search and selection process.

Table 2
Research questions.

Research questions	Motivation
RQ1: What kind of information is taken as input for the analysis of code repositories?	To know what kind of information is analysed and their respective characteristics or approaches in code repositories.
RQ2: What techniques or methods are used for analysing code repository?	To determine which are the main techniques and methods to obtain information from code repositories.
RQ3: What information is extracted (directly) or derived (indirectly) as a result of the analysis of code repositories?	To analyse what information is extracted or derived through the analysis of code repositories.
RQ4. What kind of research has proliferated in this field?	Establish the type of research that is most frequent in this area, e.g. solution proposal, applied research, research evaluation, etc., in order to know the maturity of the area and identify gaps.
RQ5. Are both academia and industry interested in this field?	To analyse the degree of interest of industry in this field through its participation in research work.

3.1. *Definition Phase*

In this phase, we define a set of activities for SMS which are the following: research questions, search process, study selection procedure, quality assessment, data extraction and taxonomy and collection methods.

3.1.1. *Research Questions*

The main research question (RQ), described in the previous section along with the main goal of our SMS, is to discover and evaluate recent published studies on the methods and techniques used for information analysis of code repositories in different digital libraries. We segment our main research question into more specific research questions in order to cover the wide scope of our main research question. Table 2 shows these research questions, together with their motivation. '*The question definition is mostly based on the grounded theory methodology which involves the construction of theory through systematic gathering and analysis of data*' (Stol *et al.*, 2016).

RQ1 focuses on the input, that is, the information taken for the analysis of the code repository. The method or technique used for the analysis of information from the code repository is then analysed through RQ2. Finally, the purpose and output produced by the analysis is finally investigated by means of RQ3., i.e. the information extracted or derived

Table 3
Main terms and synonyms or alternative terms.

Main terms	AND expression division	Alternative terms
Code repository	Conceptual synonyms	software repository
		version control systems
	Technological synonyms	Git
		Svn
Analysis	Synonyms	Mining
		Inspection
		Exploring

from the analysis. In addition, questions RQ4 and RQ5 describe the characteristics of the study. RQ4 delimits the type of research, for example, whether it is applicable or proposes a solution; RQ5 determines the involvement of the researchers who have conducted the study, for example, if the researchers are from academia or industry. Once the RQs of our study have been formulated (see Table 2), the following subsections describe the search process, study selection procedure, quality assessment, data extraction and taxonomy and collection methods.

3.1.2. *Search Process*

In a systematic mapping, an important step is to define the search process for primary studies. These studies are identified by using searches in scientific bibliographies or by browsing the research of specific known journals and conferences in the area. In our systematic mapping, we search five digital scientific databases considered relevant to software engineering recommended by Kuhrmann *et al.* (2017) for primary studies: Scopus, IEEE Xplore, ACM Digital Library, ScienceDirect and ISI Web of Science. The use of these libraries allows us to find the largest number of primary studies related to the research questions.

After selecting the scientific libraries for the search, the next step is to create the search string. We define two main terms: "Code repository" and "Analysis", to cover the terms (input and output) that are identified in the black box system (Perez-castillo *et al.*, 2019). In addition, we used the term version control system to include it in the search string because it goes similar with the main term and is a part of code repository (Dias de Moura *et al.*, 2014). Similarly, as technological synonyms, we include the terms Git and SVN because of their wide adoption and use as the most popular tools (Just *et al.*, 2016). With the main terms defined, we chose to specify some synonyms and alternative terms (see Table 3). To link the main defined terms we use AND, and to link the alternative terms we use OR. We found (test search string) that using this combination of terms we get the most studies that are a part of our approach (e.g. GitHub, GitLab, StackOverflow). The generated search string is as follows:

```
("code repository" OR "software repository" OR
"version control system" OR git OR svn) AND
(analysis OR inspection OR mining OR exploring)
```

Table 4
Inclusion and exclusion criteria.

Id	Criteria
IC1	Peer reviewed paper, for example, proceeding chapters, book chapters, keynote abstracts, call for papers and irrelevant publications.
IC2	The study employs some kind of techniques or methods to extract information through the analysis of code repositories.
IC3	The study provides some idea or type of application that might be applied for the analysis of code repositories.
IC4	The papers that were published from 1 January 2012 to 31 August 2019
EC1	The paper is duplicate
EC2	Non-English articles
EC3	The paper is a preliminary investigation which is extended or is dealt with in depth in a more recent paper by the same authors which have already been included.
EC4	The focus of the article is not within the computer science area.

We search each of the five academic databases using the defined search string, with the exception of the ISI Web of Science, which does not allow it, and therefore we apply the search string only to the title, abstract and keywords. The search string was modified for each digital library. For replication of the study, Appendix A shows each library together with the search string with the syntax needed to be used in the digital library. An important point is that a filter was made by considering only the studies from 2012 to 2019, so one of the exclusion criteria was met. This selected period (seven years) is a reasonable time period to avoid the selection of outdated, general or extensive works (Tahir *et al.*, 2013; Cosentino *et al.*, 2017). Also, it helps to avoid works in fashion peaks, that is, works in a very short period (De Farias *et al.*, 2016).

3.1.3. *Selection of Primary Studies Procedure*

The results obtained in the search in digital scientific libraries contain studies that contribute to our research and others that are irrelevant, so it is necessary to define both selection and exclusion criteria to filter those results. The practices and strategies of inclusion and exclusion of studies are valuable for the Petersen and Gencel systematic reviews (Petersen and Gencel, 2013). We defined the following criteria:

The inclusion criteria (IC1 in Table 4) refer first to the studies (IC1) that analyse the code repositories, extract information from the code repositories, using techniques and methods, what type of information is extracted and what this retrieved information is used for. The second inclusion criterion (IC2) refers to studies that propose innovative ideas or techniques that can be adapted or modified to apply to the analysis of code repositories.

Exclusion criteria (ECn) refers to the common exclusion criteria widely used in SMS to exclude, for example, duplicate papers, papers written in a language other than English, studies that present lectures and presentations, or papers that present research in other subject areas that do not meet the research area. We use the following procedure to select primary studies:

Step 1. Paper is not a duplicate.

Step 2. Apply the exclusion/inclusion criteria to the studies obtained by using the search string, along with the analysis of the title, keywords and abstract of the article containing information related to our research topic. Therefore, we included studies that met at least one of the criteria (see Table 4 for inclusion criteria). In case of doubt, we proceeded to include the document for further analysis in Step 3.

Step 3. In order to perform a more exhaustive filtering and to know which studies should be excluded or selected, we proceed to read the entire study using the exclusion/inclusion criteria. The first author was responsible for selecting the studies. In this step, selection issues were resolved by agreement among all authors after analysing the full text. We obtained the primary studies that we used for our analysis and that allow us to answer the questions posed.

After Step 2, we use another procedure to mitigate subjectivity. The remaining authors carried out the verification of the results of the study selection separately. The authors took a random sample for Step 2 and the inclusion/exclusion criteria were applied. Once the procedure was completed, the researchers checked the agreements on the selection and classification procedure of the selected studies.

3.1.4. *Quality Assessment*

In order to provide the quality assessment of the selected studies, Petersen *et al.* (2015), Kitchenham *et al.* (2013) propose criteria to perform a quality assessment for SMS in software engineering. The highest score obtained from a study means that the results are clear, with replicable results, its limitations have been analysed and its presentation is clear. In a similar way, we used these parameters to assess the quality of publications related to code repositories. An instrument with questions and a five-point rating scale was designed to determine the quality of the primary studies.

This analysis contains five subjective closed-ended and two-point objective questions. The assessment scale considers a range from 1 to 5 in quantitative terms, i.e. based on the Likert-scale (Pedreira *et al.*, 2015). The possible answers to these questions show the reviewer's level of agreement, and range between 1 = "Strongly disagree" 2 = "Disagree", 3 = "Neither agree nor disagree", 4 = "Agree", 5 = "Strongly agree". In order to carry out the evaluation of the selected papers, considering subjectivity, group discussion sessions were held with other experts, so that the assessment of each evaluation question for each paper was obtained by consensus and independently.

The quality assessment provided us with guidelines and aspects related to research in area of code repositories and the information that is entered, processed, analysed, and is used in different aspects. Table 5 presents the questions of the instrument used. (AQ1) evaluates the primary studies in relation to the analysis of information from code repositories (a systematic approach); (AQ2) if the study presents a result of the analysis of information from code repositories; (AQ3) if the study uses an artifact (method, technique, tool) for the processing of information from code repositories; (AQ4) if the study provides a solution to the problems of quality, development and evolution of software or not. (AQ5) if the research provides any artifact (method, technique, tool) that can be applied in an industrial environment (see Table 5). (AQ6) estimates the number of citations, which we

Table 5
Quality assessment questions.

Nr.	Assessment questions	Criteria
AQ1	Does the study have a systematic method for obtaining baseline information for code repository analysis?	Defined methods
AQ2	Does the study present a result of code repository information analysis?	Data analysis
AQ3	Does the study present an artifact (technique, tool, or method) for processing information from code repositories?	Study presentation results
AQ4	Does the research show a solution to the problems of software quality, development and evolution?	Study focus
AQ5	Does the research provide an artifact (technique, tool or method) that can be applied in industrial environments?	Application
AQ6	Do other authors cite the selected study?	Utility
AQ7	Is the journal or conference that publishes the study important or relevant?	Relevant

obtained from the various digital scientific databases. For AQ6, the scale values we use are the value of '1' for the score of the studies with the least amount and the value of '5' with the studies with the most amount of citations. In addition, we standardized the papers, dividing the number of citations by the number of total years published. The standardization of papers helps us to avoid penalties for recent publications. AQ7 determines whether the conference or journal publishing the study is outstanding or important. To measure this question, we considered the relevance index collected by two conference classifications: CORE ERA and Qualis. These conferences were standardized with ranges ('A', 'B', 'C') for the first one and ('A1', 'A2', 'B1', 'B2', 'B3', 'B4', 'B5') for the second one to finally obtain a calculated average. In the case of journal articles, we rely on the Journal Citation Reports (JCR) quartiles, which have their index ('Q1' = '5', 'Q2' = '4', 'Q3' = '3', 'Q4' = '2', 'Q5' = '1') that is in a descending order with the lowest '1' representing non-indexed journals.

3.1.5. *Procedure for Data Extraction and Taxonomy*

We obtained clear and systematic information using a data extraction instrument (see Appendix B). We defined the possible answers for research questions posed in the previous sections (see Table 2). We obtain a homogeneous cluster by extracting the criteria from the selected studies and allowing for a taxonomy. For taxonomy we take as a basis the taxonomy provided (Dit *et al.*, 2013), other similar studies (Kagdi *et al.*, 2007; Cavalcanti *et al.*, 2014) and our pilot study. Main category consists of a set subcategory that shares common characteristics and type quantitative. For example, the category "Empirical/Experimental" in Table 6, which corresponds to RQ3, is grouped together with research that employs methods and techniques, mentioned bellow: empirical study, empirical evaluation, controlled experiment, experimental study case, study empirical analyses, exploratory study. Extraction procedure was tested using the form in a pilot study (see Appendix B). Intention of the pilot studies is "to evaluate both technical issues, such as the completeness of the forms, and usability issues, such as the clarity of the instructions for use and the order of the questions" (Kitchenham, 2007). The process of item categorization was carried out

Table 6
Taxonomy.

RQs	Categories
RQ1	Project Features Info
	Defects
	Comments
	Branches
	Source Code
	Informal Information
	Committers
	Commit Data
	Logs
	Graphs/News Feed
	Issue
	Pulls/Pull Request
	Level of Interest
	Repository Info
RQ2	Automatic Processing
	Branching Analysis
	Changes Analysis
	Commits/Committers Classification
	Cloning Detection
	Code Review
	Commit Analysis
	Defect/Issues Analysis
	Developer Behaviour
	Design Modelling
	Maintainability Information
	Metrics/Quality
	Source Code Improvements
	Testing Data
RQ3	Ad Hoc Algorithms
	Data Mining
	Automatic
	Artificial Intelligence/Machine Learning
	Qualitative Analyses
	Heuristic Techniques
	Empirical/Experimental
	Statistical Analyses
	Prediction
	Reverse Engineering
	Testing-Based Techniques
RQ4	Evaluation Research
	Proposal of Solution
	Validation Research
	Philosophical Papers
	Opinion Papers
	Personal Experience Papers
RQ5	Industry
	Academia
	Freelance

by the authors individually. Items with some disagreements were identified and a discussion table was held about them. The set of attributes was extracted and defined by the authors. The characterization of the articles by the authors allows us to verify the quality of the taxonomy, minimising possible bias. At the discussion table, disagreements served as a parameter that our taxonomy and content needed to be refined. Table 6 shows in more detail the taxonomy we developed for each research question.

3.1.6. *Summary Methods*

We summarize the results through both qualitative and quantitative approaches. The qualitative approaches are as follows:

- Quality assessment is an important parameter when selecting studies according to research questions.
- We delimit the research questions with a classification and a quality evaluation.

The quantitative approaches are as follows:

- We generate a taxonomy of the selected studies according to each research question (see Table 6).
- We made a summary with the total number of articles per country and per year (see Fig. 3).
- We prepared a matrix of each primary study distributed in rows containing information on the research questions, proposed taxonomy and quality assessment.
- To summarize the results of the SMS, we generated a bubble chart where the different research questions intersect with the number of the selected primary studies.

According to Petersen and others (Petersen *et al.*, 2008), a bubble plot "is basically two $x - y$ scatter plots with bubbles at category intersections". The proportion of the bubble size depends on the number of studies that are distributed in the $(x - y)$ categories of the bubble.

3.2. *Execution Phase*

The execution of our SMS was carried out by three researchers, with time of 9 months to finish. The systematic mapping study schedule began with protocol development and improvement, extraction, and elimination of duplicates. This was followed by study selection by analysing the title, abstract and keywords. Another iteration applied the inclusion and exclusion criteria. Then, all the primary studies selected in this step were downloaded. The selection process is determined by the full text, we apply taxonomy, classification, and quality assessment. Conflict resolution is carried out by focus group sessions. And finally, report of all the steps executed and the activities carried out throughout the study was generated. In Figure 2 we can see a summary of the search and selection process of primary studies and their respective results.

Altogether, we obtained 3755 publications as a result of the automatic search in the digital libraries. As a first step, we eliminated duplicates (502 studies) obtaining a total of

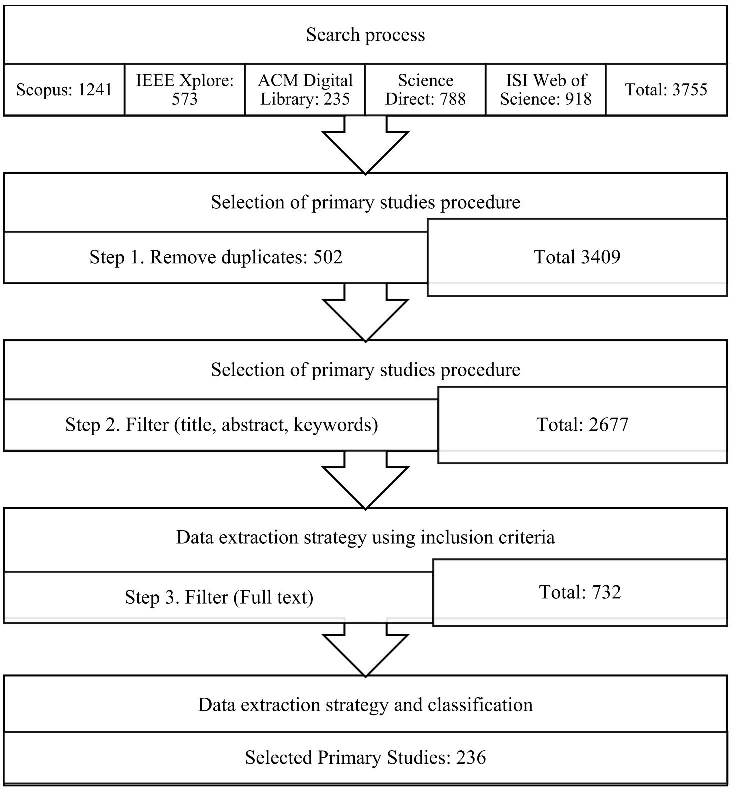
Search process
Scopus: 1241
IEEE Xplore: 573
ACM Digital Library: 235
Science Direct: 788
ISI Web of Science: 918
Total: 3755
Selection of primary studies procedure
Step 1. Remove duplicates: 502
Total 3409
Selection of primary studies procedure
Step 2. Filter (title, abstract, keywords)
Total: 2677
Data extraction strategy using inclusion criteria
Step 3. Filter (Full text)
Total: 732
Data extraction strategy and classification
Selected Primary Studies: 236

Fig. 2. Results obtained from the search and selection process.

3409 studies. Then, applying the exclusion and inclusion criteria, we selected 732 publications. As the last one that corresponds to the complete reading of the study, we selected 236 studies. Appendix C shows the list of the primary studies we selected. We subsequently conducted the extraction of data, classification and synthesis with these 236 primary studies. It is possible to view the tables obtained from the data extraction, classification and synthesis online at https://GitHub.com/jaimepsayago/SMS.

4. Results of the Systematic Mapping Study

The search process was carried out by following the criteria and strategies described in the previous section. Figure 3 and Appendix A show a summary of the number of papers obtained in each step of the search process regarding the year in which the primary studies were published and which country their authors were from, respectively.

According to the results shown in Fig. 3, the number of primary studies obtained may appear to be large. We considered studies published between 2012 and 2019 for the reasons explained in Section 3.1.3. The distribution shows an upward trend regarding the papers

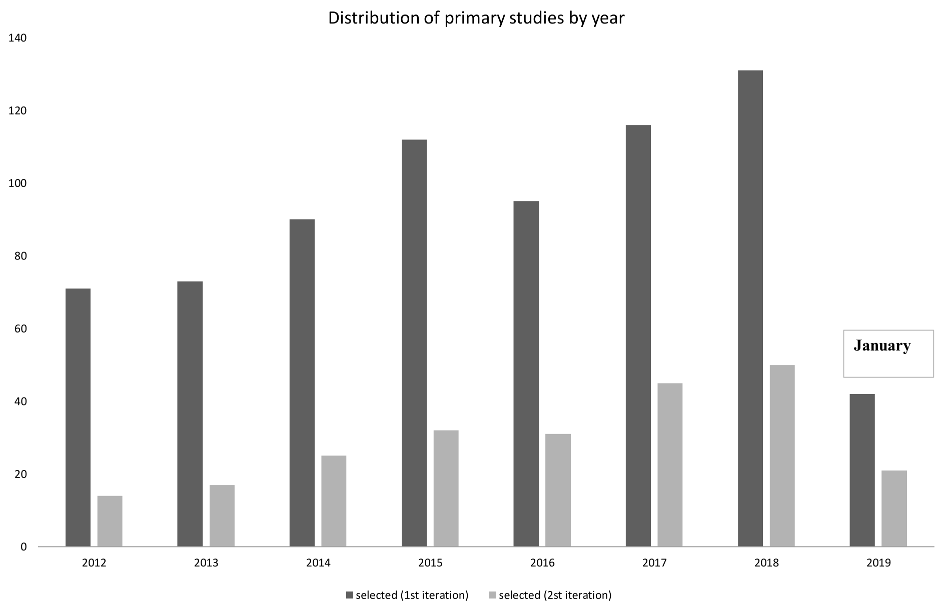

Fig. 3. Distribution of primary studies by year.

retrieved from digital libraries in the most recent years. The first primary studies focusing on code repositories were published at the beginning of the 2000s. The number of studies published in 2012 are on par with those of 2013. The amount of studies is much higher in 2014 and 2015. In this sense, the number of primary studies published in 2016 is lower than in the previous years. Nevertheless, there is a spike in the number of articles published in 2017 and 2018. In the year 2019, the number of studies decreases because of the cut in our mapping: the search lasted from January to August, although the trend of papers for the year 2019 is high. This result seems to follow the trend even though we do not complete the whole year 2019. Yet we can see a growing interest from researchers in code repositories.

Figure 4 shows the distribution of the studies (we include affiliation and location of each of the authors) according to the year they were published. It reveals that most of the selected papers come from the American continent, the first is USA (17%), followed by Brazil (10%), Canada (10%) and then China (9%), Japan (9%) and India (8%). Despite these top countries, code repository analysis is widely studied around the world, demonstrating the importance of the topic.

In terms of the type of publication, the studies were published as conference proceedings with 60% and conference papers with 10%, respectively (see Fig. 5). Journal articles represented only 23% of the total selected primary studies. The 4% correspond to series and 3% to book sections. By analysing these results, we can observe that there are certain efforts to achieve a greater maturity in this field of research. However, it must be taken into account that this is a field that has been intensively researched during the last decade.

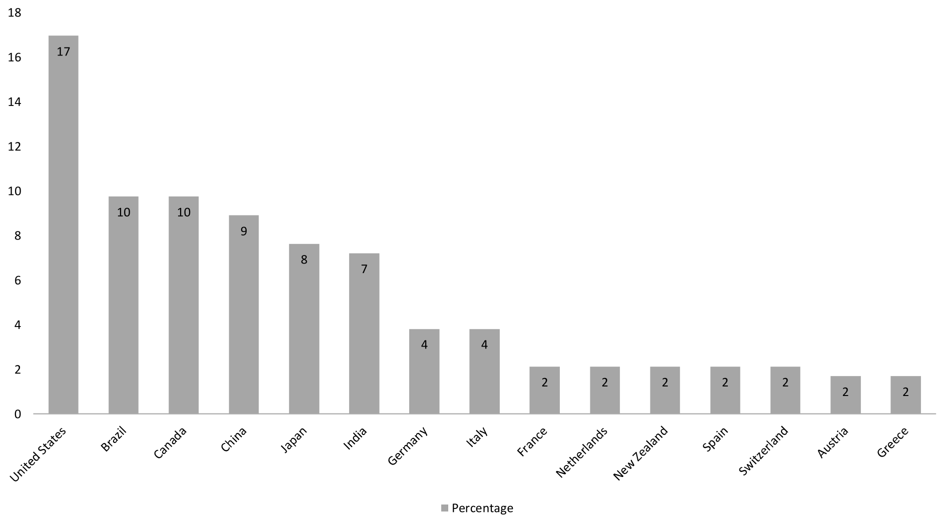

Fig. 4. Distribution of primary studies by country.

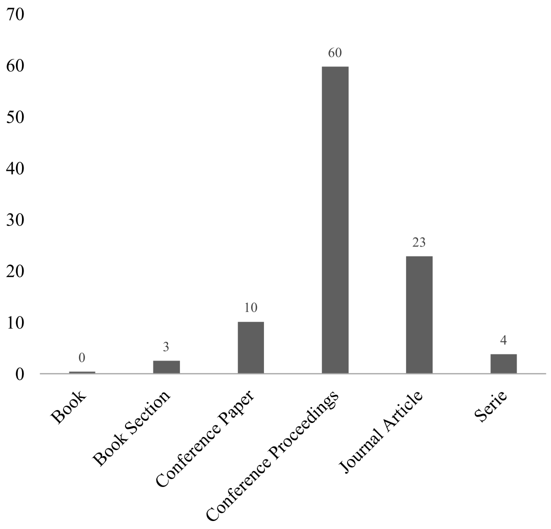

Fig. 5. Types of publications considered in research.

In this section an analysis is performed with the 236 primary studies obtained following the classification criteria and research questions that have been previously outlined (see Table 2 and 6). The answers to the stated research questions, according to the analysis performed on the primary studies selected are depicted in next sections.

4.1. *RQ1. What Kind of Information is Taken as Input for the Analysis of Code Repositories?*

Table 7 shows the classification of categories made for the first research question. The selected studies exhibit different artifacts that require some kind of grouping. For this purpose, we have created a taxonomy for code repositories. For this taxonomy, we take as a basis the taxonomy provided in Dit *et al.* (2013) as well as other similar studies (Kagdi *et al.*, 2007; Cavalcanti *et al.*, 2014). The 14 possible inputs considered for RQ1 are grouped and evaluated as shown in Table 7.

Table 7 shows how the selected primary studies are distributed in relation to (RQ1), the number of studies for each category and the percentage distribution. We observed that the proposed distribution includes the different components present in a code repository (commits, pulls, branches, etc.). We identified several studies that combine more than one source of data to achieve their analysis of the code repository but do not reach a significant number within our research (less than 2%). For the analysis of code repositories are the commits; commit messages is the most recurrent input employed in selected studies, with 115 studies in total (34%). This means the type of information most used as input, contributors, commit history, etc. In particular, the most relevant studies of this category focus on the use of repository commits that record changes in the source code made by developers.

Some of the studies in this category focus on taking as the main information the commitments to follow up on issues that may occur in the project or with the developers, and that represent a challenge when executing software maintenance. Thus, the study 115 (Jarczyk *et al.*, 2017), the closing of issues (errors and characteristics) is studied using it as main information for the analysis (commits) in software projects, which allows a better understanding of the factors that affect the completion rates of issues in software projects. As for example in the study 130 (Kagdi1 *et al.*, 2014), the authors propose to use commits in repositories that record changes to source code submitted by developers to version control systems. This approach consists of recommending a classified list of expert developers to assist in the execution of software change requests.

The second most common entry with 90 studies (26%) (see Table 7) is the input "source code" which represents a huge body of software and related information for researchers who are interested in analysing different properties related to the source code of software. Specifically, source code allows a meaningful understanding of software development artifacts and processes (Dyer *et al.*, 2015). For example, the study 187 (Negara *et al.*, 2014) provides an approach that previously identifies unknown frequent code change patterns of a sequence of fine-grained code changes and that allows understanding the evolution of the code.

Other studies in this category also focus on analysing the repository code to investigate the development process. The study 80 (Finlay *et al.*, 2014) takes as main information (input) the code and comments to obtain metrics to relate them to the development effort.

The third most common entry is the input "information repository" with 23 studies (7%) (see Table 7), it mainly groups historical data, dataset repository and historical code

Table 7
Classification of selected papers by the input used (RQ1).

Input	Papers	# studies	%
Commit data	62, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 185, 189, 203, 223, 225, 226, 227, 229, 230, 233, 235	115	34
Source code	7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 167, 176, 203, 224, 226, 228, 229, 230, 235, 236	90	26
Repository info	61, 179, 180, 181, 182, 190, 191, 192, 193, 194, 195, 213, 214, 215, 216, 217, 218, 219, 220, 221, 222, 232, 234	23	7
Issue	84, 85, 166, 167, 173, 174, 175, 178, 195, 196, 197, 198, 199, 200, 201, 202, 225, 229, 234	19	6
Comments	63, 64, 65, 66, 87, 88, 89, 90, 91, 157, 158, 159, 185, 198, 199, 200	16	5
Branches	1, 2, 3, 147, 148, 149, 165, 177, 185, 186, 187, 188, 189	13	4
Defects	4, 5, 6, 80, 81, 82, 150, 151, 152, 153, 154, 155, 233	13	4
Pulls/pulls request	146, 172, 173, 174, 175, 205, 206, 207, 208, 209, 210, 223, 225	13	4
Commiters	83, 102, 153, 154, 155, 160, 161, 162, 163, 164, 165, 233	12	4
Informal information	86, 156, 168, 169, 170, 171, 203, 204	8	2
Level of interest	177, 188, 189, 211, 212, 223	6	2
Proyect features info	61, 62, 86, 100, 101, 189	6	2
Logs	6, 144, 145, 183, 184	5	1
Graphs/news feed	201, 202	2	1

repository. These studies focus on obtaining the general information of the repository order to obtain useful knowledge for the development and maintenance of the software.

In this category, we found the study 232 (Wu *et al.*, 2014) that takes the information from the code repository to analyse social characteristics of collaboration between developers. The authors focus on demonstrating that code repositories are a part of a broad ecosystem of developer interactions. Another example is the study 191 (Novielli *et al.*, 2018) that takes the information from the code repository to analyse the emotions of developers, applying sentiment analysis to the content of communication traces left in collaborative development environments.

The remaining categories are as follows. The next input is the “issues” category with 19 studies (6%) (see Table 7), the issues are processed for the resolution of the problem or question of something specific. The fifth input is the “comments” category with 16 studies (5%) which can be seen as an important complementary analysis component in a repository. These five categories are the most common inputs for this classification.

Other studies employ alternative inputs that are used in groups of 12 to 13 studies, representing less than 5% (see Table 7). Those categories are “branches”, “defects”, and “pulls and pull requests” and “committers”. The studies take characteristics from code repositories as mentioned in Section 2.1. In study 75 (Elsen, 2013) mention that a potential contributor may participate in the development or maintenance process by submitting a pull request, which will be reviewed for acceptance or rejection by the central development team. It is here, that in addition to hosting software repositories, features such as "defects" of the developers and the "branches" or "forks" of the projects are incorporated into the development process (Liu *et al.*, 2016). These actions and interactions of the developers are to be collected and allows the possibility of analysis within the code repositories.

Other five categories with 27 studies in total represent no more than 8% (see Table 7) of the total studies in RQ1. These studies are directed at features of the code repositories presented in Section 2.1. The categorization with the type of information as input is "informal information" which focuses on "chats", "mails" and "messages" interchanged. "Level of interest" relates to the social components of the project such as "stars", "follows" and "watches", that are mechanisms typically offered in public open source repositories like, for example, GitHub. "Project features info" involves aspects like the "size", "owner", "weeks", "contributors" which are a part of the general features of the code repository. "Logs" and "graphs/news feed" are categories that also contribute as inputs to the analysis of the code repository.

4.2. *RQ2. What Techniques or Methods are Used for Analysing Source Code Repository?*

The methods or techniques used in the process of analysis of the code repository can be observed in Table 8 together with the distribution of studies for this question. Results for this question were obtained using the procedure for data extraction and taxonomy provided in Section 3.1.5. There are some papers that are present in more than one category, i.e. different methods contributed by the paper were counted. As a result, the percentage column in Table 8 represents the total.

The mapping indicates that the most common is "empirical/experimental" with 93 studies (38%) (see Table 8). This type of studies is a systematic, disciplined, quantifiable and controlled way to evaluate information and approaches against other existing ones and to know under which criteria they are better (Genero Bocco *et al.*, 2014). These methods include empirical studies, empirical evaluations, experimental studies, empirical analyses, case studies, systematic literature reviews, research strategy, etc.

The second most recurrent methods are tagged as "automatic" with 29 studies (12%) (see Table 8).These studies focus on using tool automation techniques to perform a specific task. For example, 67 (Dias *et al.*, 2015) proposes an automatic tool to untangle fine grain code changes in groups, allowing good results with an average success rate of 91%. The proposal of study 176 (Martínez-Torres *et al.*, 2013) develops an automatic categorization tool to extract text for the analysis of knowledge sharing activities in projects.

The third input is "artificial intelligence/machine learning" (AI/ML) with 29 studies (12%) (see Table 8). These studies use techniques that are relevant today as they have achieved a remarkable momentum that, if properly used, can meet the best expectations in many application sectors across the research field (Barredo Arrieta *et al.*, 2020). Due to this importance, we provide a specific classification within this category.

Table 8
Classification of selected papers by concern/topic, number or studies and percentage (RQ2).

Methods-techniques	Papers	# studies	%
Empirical/experimental	2, 5, 8, 16, 17, 18, 19, 22, 23, 24, 25, 26, 27, 28, 30, 31, 32, 34, 35, 36, 42, 43, 44, 45, 46, 48, 52, 56, 57, 59, 60, 62, 63, 64, 65, 66, 68, 74, 77, 83, 88, 91, 97, 100, 101, 102, 107, 111, 113, 114, 118, 120, 121, 128, 138, 139, 143, 147, 148, 151, 154, 161, 163, 165, 169, 177, 179, 184, 185, 186, 189, 192, 193, 194, 196, 197, 202, 206, 209, 211, 212, 213, 215, 216, 220, 222, 223, 225, 226, 227, 229, 233, 235	93	39
Automatic	14, 51, 54, 55, 67, 71, 75, 76, 78, 81, 82, 85, 90, 96, 99, 105, 109, 116, 131, 153, 159, 168, 171, 174, 176, 181, 208, 219, 230	29	12
Artificial intelligence/machine learning	1, 3, 4, 33, 49, 50, 53, 69, 70, 80, 87, 106, 108, 122, 126, 127, 130, 142, 150, 157, 158, 166, 175, 178, 182, 203, 205, 207, 214	29	12
Statistical analyses	6, 9, 12, 21, 37, 38, 39, 58, 86, 89, 94, 98, 112, 136, 167, 199, 201, 204, 210, 217, 228, 232, 234, 236	24	10
Ad hoc algorithms	20, 29, 41, 47, 61, 63, 72, 92, 103, 104, 115, 129, 137, 140, 146, 149, 162, 166, 173, 183, 187, 188, 195	23	10
Data mining	7, 15, 53, 95, 110, 132, 141, 145, 152, 157, 164, 200, 217, 221, 231	15	6
Qualitative analyses	13, 125, 127, 144, 158, 160, 191, 218, 224	9	4
Prediction	10, 53, 134, 157, 190, 198, 203, 214	8	3
Reverse engineering	11, 73, 84, 133, 155, 180	6	3
Heuristical techniques	93, 119, 123, 124, 156, 172	6	3
Testing-based techniques	40, 79, 117, 135	4	2

The vertiginous increase of artifacts using AI/ML techniques demonstrates the inclination of the software engineering community towards this branch. These are not isolated cases or fads (Harman, 2012). Nowadays, the nature of software goes hand in hand with human intelligence; this is where AI/ML techniques are becoming a part of software, specifically in the field of code repositories.

The renewed interest and number of AI/ML techniques has led to many advances related to this field. For example, Bayesian statistics (Abdeen *et al.*, 2015), Convolutional Neural Network (Li *et al.*, 2019) and Random Forest Classifier (Maqsood *et al.*, 2017). These are used to understand bugs or make predictions of possible code changes, finding and predicting defects in a code repository or identifying code repository errors. Besides, it has been criticised that many of these approaches to building smarter software are too far from human-level intelligence and are therefore likely to be insufficient (Feldt *et al.*, 2018). This situation has focused on the need for less complex algorithms and tools to be integrated into the systems and solutions that are used by organizations.

Table 9 shows the taxonomic subcategories that we have carried out based on (Baltrusaitis *et al.*, 2019) and (Agarwal *et al.*, 2019) for "Machine Learning" and (Gani *et al.*, 2016) for "Artificial Intelligence". These techniques and methods aim to build models that can process and relate information from multiple sources. It is worth mentioning that these techniques have a growing importance and an extraordinary potential.

There exist examples where AI/ML models are applied to improve software development, specifically in the area of code repositories. For example, the study 87 (Fu *et al.*, 2015) uses the technique Latent Dirichlet Allocation (LDA) to extract information from

Table 9
Components of the category artificial intelligence/machine learning.

Artificial intelligence/machine learning	Papers	# studies	%
Random Forest Classifier	49, 50, 157, 175	4	14
Natural Language Processing (NLP)	126, 130, 178	3	10
Bayesian classifier	3, 205	2	7
Search-based genetic algorithm	33, 207	2	7
Latent Dirichlet Allocation (LDA)	87, 106	2	7
Naive Bayes-based approach	122, 166	2	7
Artificial Intelligence	4, 70	2	7
Statistical learner	203, 214	2	7
Sentiments analysis tools	69	1	3
Deep model structur (convolutional Neural Network)	158	1	3
Rule-based technique	1	1	3
semantics-based methodology	150	1	3
SGDClassifer	142	1	3
Machine learning techniques	127	1	3
Hoeffding tree classification method	80	1	3
Dynamic topic models	108	1	3
Naive bayes classifier	182	1	3
Gradient boosting machine	157	1	3

the change messages of the repository to classify them in an automatic way. Another example is the study 127 (Joblin *et al.*, 2015) where a general approach is proposed for the automatic building of developer networks based on source code structure and commit information, obtained from a code repository that is applicable to a wide variety of software projects.

Other examples are the study 122 (Jiang *et al.*, 2019) that uses a random forest classifier and naive bayes classifier together with the study 3 (Abdeen *et al.*, 2015) that uses a Bayesian classifier. Both studies use those classifiers as the main technique to process and analyse the input information and generate models to predict different aspects of the code repositories (change impact or code review, among others).

The proposed taxonomy aids the understanding and comprehension of AI/ML techniques used in code repository analysis.

Continuing with the taxonomy for (RQ2), other relevant techniques used in the selected studies are those related to "statistical analyses" appearing with 24 studies (10%) (see Table 8), this category groups different techniques such as "Micro-Productivity Profiles Method", "Quantitative analysis", "Models regression", "Regression tree", etc. Some examples of the application of these techniques are studies 89 (Gamalielsson and Lundell, 2014) through a review quantitative analysis of project repository data in order to investigate the sustainability in OSS communities with a detailed analysis of developer communities, the authors of study 86 (Foucault *et al.*, 2015) provide a quantitative analysis of the rotation patterns and effects of developer that along with the activity of external newcomers, affect negatively the quality of the software; or the study 38 (Borges and Tulio Valente, 2018) provides strong empirical quantitative evidence about the meaning of the number of stars in the code repository, recommending to monitor this metric to use it as a pattern for repository selection.

Following the classification we find the use of “ad hoc algorithms”, that are used in 23 studies, representing 9% (see Table 8). In these studies, specific algorithms are provided, for example, semantic slicing, gumtree, prediction partial matching, etc. These are applied, for example, for information analysis, as in the study 61 (Datta *et al.*, 2012) algorithms to determine social collaboration teams are employed. Also, the study 173 (Malheiros *et al.*, 2012) provided an algorithm to analyse change requests and recommend potentially relevant source code that will help the developer.

Another category is “data mining” with 15 studies (6%) (see Table 8). Data mining refers to the extraction or “mining” of knowledge from large volumes of data (Grossi *et al.*, 2017). Studies rely on these techniques (“Hierarchical agglomerative clustering”, “Information retrieval (IR)”, “Decision Tree”, “C4.5”, “Logistic Regression”, “k-Nearest Neighbour (k-NN)”, etc.) to process data from code repositories.

The rest of the studies add up to 15% of RQ2. Qualitative analyses with 9 studies (4%). “prediction” with 8 studies (3%), “reverse engineering with 6 studies (2%). “heuristical techniques” with 6 studies (2%). Finally, “testing-based techniques” with 4 studies (2%) (see Table 8).

4.3. *RQ3. What Information is Extracted (Directly) or Derived (Indirectly) as a Result of the Analysis of Source Code Repositories?*

Having analysed the studies according to RQ3, the main output generated is information related to “developer behaviour” with 65 studies (26%) (see Table 10). Currently researchers have been motivated by the lack of research on developer-related social processes oriented to management, analysis, maintenance and teamwork (Gamalielsson and Lundell, 2014) and we see this is reflected in the mapping study. The category groups different characteristics of the developer that can be extracted from the code repositories, for example, with the purpose of knowing developers’ patterns, developers’ sentiment classification, developer contribution analysis, developer social networks, development processes, etc. These outputs are eventually used to improve the maintenance and generally to comprehend the evolution of software.

For example, we can point out the study 186 (Murgia *et al.*, 2014) where emotion mining is performed, applied to developers’ problem reports, and it can be useful to identify and monitor the mood of the development team, which allows to anticipate and solve possible threats in their team. Another example is the study 130 (Kagdi1 *et al.*, 2014) that proposes an approach to recommend a classified list of expert developers to assist in the implementation of software change requests (e.g. bug reports and feature requests).

The second most recurrent output is “changes analysis” with 35 studies (14%) (see Table 10). These studies are interesting since the information extracted from these code changes can be used to predict future defects, analyse who should be assigned a particular task, obtain information on specific projects or measure the impact of the organizational structure on software quality (Herzig *et al.*, 2016).

For example, the study 138 (Kirinuki *et al.*, 2014) proposes a technique to prevent “tangled changes” in which it is identified whether a developer’s changes are tangled and

Table 10
Classification of selected papers by concern/topic, studies and percentage (RQ3).

Output	Papers	# studies	%
Developer Behaviour	1, 5, 8, 11, 13, 17, 21, 26, 30, 36, 37, 38, 39, 40, 43, 48, 52, 55, 56, 60, 61, 64, 65, 66, 69, 71, 77, 89, 92, 99, 101, 116, 120, 121, 127, 130, 147, 151, 160, 161, 163, 169, 175, 177, 181, 186, 189, 190, 191, 192, 196, 200, 204, 208, 211, 215, 216, 220, 221, 222, 230, 231, 232, 233, 234	65	26
Changes Analysis	3, 4, 31, 32, 33, 42, 45, 51, 59, 63, 67, 70, 102, 104, 107, 108, 114, 118, 124, 136, 138, 155, 162, 171, 174, 179, 180, 184, 187, 194, 195, 217, 223, 229, 235	35	14
Metrics/Quality	23, 29, 58, 74, 78, 79, 80, 84, 90, 91, 105, 115, 128, 131, 145, 153, 164, 166, 168, 170, 185, 198, 199, 228	24	10
Deffect/Issue Analysis	10, 12, 15, 16, 27, 34, 41, 44, 84, 97, 113, 119, 134, 150, 167, 197, 198, 203, 205, 207, 224	21	9
Source Code Improvements	2, 9, 18, 19, 25, 49, 54, 63, 67, 72, 73, 95, 102, 110, 125, 133, 141, 164, 165, 202, 212	21	9
Commits/Committers Classification	6, 20, 46, 50, 87, 96, 98, 111, 126, 142, 146, 157, 176, 178, 182, 209, 210	17	7
Cloning Detection	22, 35, 81, 82, 85, 94, 112, 144, 149, 159, 188, 227, 236	13	5
Maintenability Information	7, 28, 57, 62, 68, 76, 83, 106, 143, 168, 226	11	4
Design Modelling	86, 88, 100, 103, 129, 148, 193, 206, 213, 218	10	4
Commit Analysis	14, 24, 47, 53, 122, 123, 137, 214, 219	9	4
Automatic Processing	109, 117, 126, 158, 172, 173, 183	7	3
Code Review	132, 139, 166, 201, 225	5	2
Branching Analysis	24, 75, 140, 152, 156	5	2
Testing Data	93, 135, 154	3	1

using the technique, developers can be made aware that their changes are potentially tangled and can be given the opportunity to commit the tangled changes separately. The study 187 (Negara *et al.*, 2014) presents an approach that identifies previously unknown frequent code change patterns of a sequence of fine-grained code changes.

The next output is tagged as "metrics/quality" with 24 studies (10%) (see Table 10). Software metrics and measurements are those processes or tools that include the assessment of the software product, project or process in order to obtain values that can help give indicators of one or more software attributes (Abuasad and Alsmadi, 1994, (2012)). This category is made up of these specific outputs like change analysis, change contracts, change histories, change impact analysis, etc. To exemplify, the study 80 (Finlay *et al.*, 2014) describes the extraction of metrics from a repository and the application of data flow mining techniques to identify useful metrics to predict the success or failure of the construction. We can also mention the study 145 (Kumar *et al.*, 2018) that proposes the creation of an effective failure prediction tool by identifying and investigating the predictive capabilities of several well-known and widely used software metrics for failure prediction.

Then, the output "deffect/issue analysis" with 21 studies (9%) (see Table 10) groups studies focusing on repository data and are employed to provide software analytics and predict where defects might appear in the future (Rosen *et al.*, 2015). An example of this is the paper 207 (Rosen *et al.*, 2015), which presents a tool that performs analysis and predicts risks in software by performing commits. Alternatively, the study 97 (Gupta *et al.*, 2014) proposes a run-time process model for the error resolution process using a process mining tool and an analysis of the performance and efficiency of the process is performed.

Another category of importance is "Source Code Improvements" with 21 studies (9%) (see Table 10), grouped according to identifiers in source code, source code legibility, annotations, source code plagiarism detection, scope of source code comments, etc.

The output "Commits/Committers" with 17 studies (7%) (see Table 10) corresponds to studies that usually extract information to perform a classification of commit messages, change messages, committers or commits from the code repository.

The following output categories in RQ3 correspond to "Cloning Detection" with 13 studies (5%) and is made up of studies where information about code cloning that aims to detect all groups of code blocks or code fragments that are functionally equivalent in a code base is derived (Nishi and Damevski, 2018). "Maintainability Information" with 11 studies (4%) is made up of studies on traceability, maintainability or technical debt. "Design Modelling" with 10 studies (4%) has studies that extract information on UML models, EMF patterns or patterns of social forking. "Commit Analysis" has 9 studies (4%), "Automatic Processing", 7 studies (3%), "Code Review", 5 studies (2%), "Branching Analysis", 5 studies (2%) and "Testing Data", 3 studies (1%) (see Table 10).

Figure 6 presents a bubble graph summarizing the combination of principal questions (RQ1, RQ2, RQ3) organized as the black box model, starting with the input, the method/technique and the output (Section 3.1.1). The largest bubble (47 studies) represents studies that take as input the category "Source Code" and the methods or techniques for processing are "Empirical/Experimental". After this, the second largest bubble (37 studies) represents that the information taken for the analysis is the "Commit Data" and the techniques, used for processing it, are equally empirical or experimental. On the other hand, the third bubble (37 studies) in terms of information extracted from the analysis of the repositories shows that it is used for "Developer Behaviour". We observe that in the bubbles of "Automatic", "Data Mining" and "Artificial Intelligence/Machine Learning" these techniques are used to process information and it has become an emerging field to process information from the repositories. Another interesting point to highlight is that all categories of both input and output use empirical/experimental techniques and methods to process information.

Analysing the data obtained from the SMS, we observe that the main trend in code repository research focuses on using empirical or experimental techniques (93 studies) in source code, code review and code repository commits to obtain results, especially related to developers' analysis (67 studies). Research trends seem to gravitate towards analyses of code changes and the impact they have on software maintenance and evolution. Analyses, metrics, measurements and classification of developers' feelings, efforts and contributions are the trends revealed by the SMS. Another marked trend in the research is the analysis of defects, issues and bugs present in the software and looking for patterns or ways to find these defects or predict them.

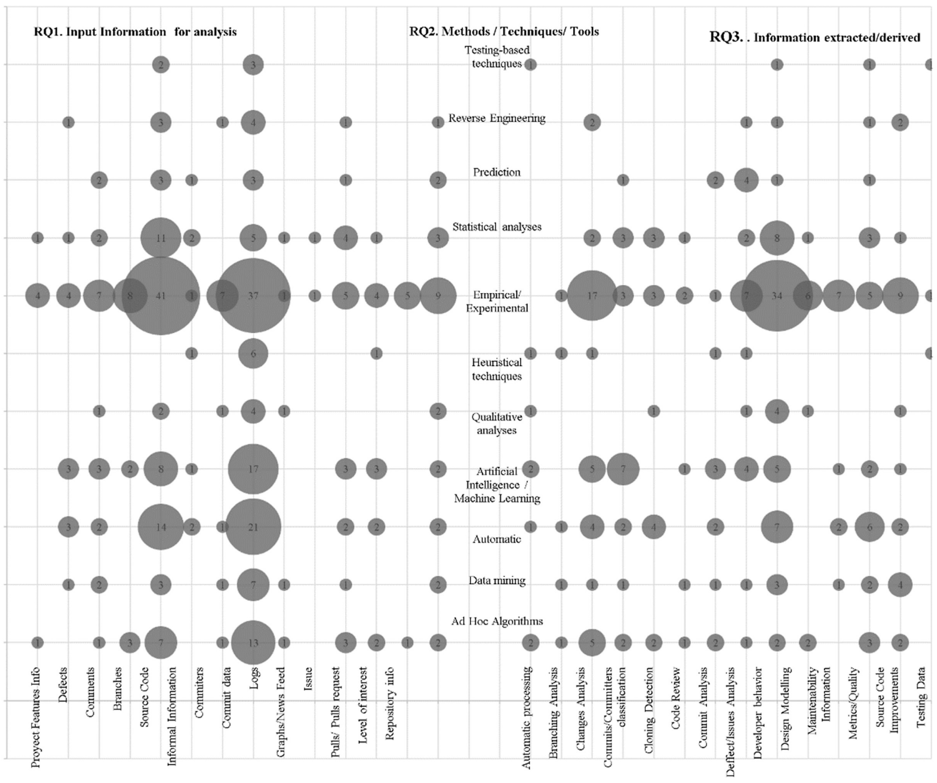

Fig. 6. Bubble graph intersecting research questions RQ1, RQ2 and RQ3.

4.4. *RQ4. What Kind of Research Has Proliferated in this Field?*

Figure 7 describes the arrangement of the primary studies according to research questions RQ4 and RQ5. The definition of the kind of contribution for each paper was done alongside the data extraction procedure (Section 3.1.5). There also may be papers that are present in more than one category to provide a solution. The main and central contribution for each paper was analysed for figuring out its classification. Regarding the nature of the research, the graph shows that the majority of studies (90%) provides solution proposals. A further 4% of studies are applied research. Two percent of studies are classified as validation research. Finally, remaining 4% are classified as evaluation research (1%), opinion articles (1%), personal experience articles (1%), philosophical articles (1%).

4.5. *RQ5. Are Both Academia and Industry Interested in this Field?*

With respect to industry interest in the field of research, Fig. 7 shows that 97% of the selected studies are authored by at least one affiliate of a university or research centre. This percentage is very high, which allows us to know that researchers are interested in

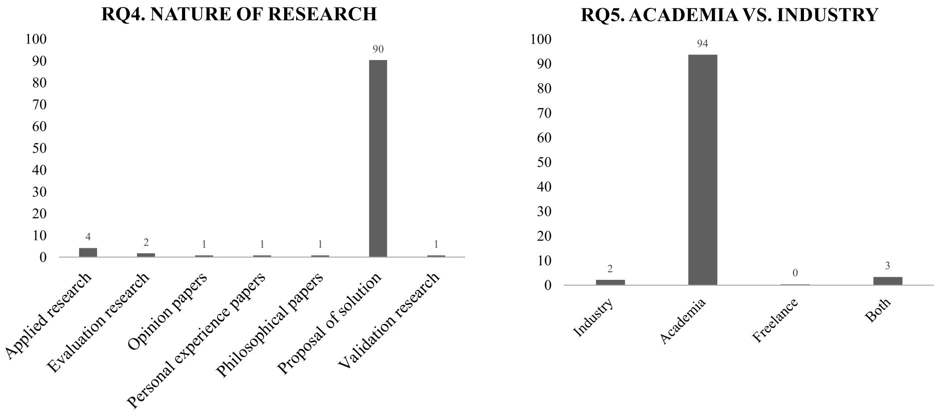

Fig. 7. Description the layout of the primary studies according to research questions RQ4 and RQ5.

the different areas of code repositories. The classification that follows is "Both" with 3%, these studies have a mixed authorship between academia and industry.

4.6. *Quality Assessment*

Finally, we used the instrument for quality assessment (Section 3.1.4) with the primary studies. Figure 8 shows the quality assessment of the seven assessment questions, in which the instrument is applied with its respective scale. AQ1 to AQ5 are questions that are evaluated in quantitative terms, while AQ6 and AQ7 are objective questions. The systematic method (AQ1) of the selected studies, which represents whether it is possible to replicate the methods and techniques systematically, resulted in the majority with high values (mostly evaluated as '5'), the other studies were rated between 3 and 4. Regarding the presentation of a result of the analysis of the code repository (AQ2), most of the studies (176) were rated as '5' (see Fig. 8). This shows that the studies present some proposal or result of the analysis carried out. In terms of methods, tools or related aspects (AQ3), most of the studies obtained a value of '5' (see Fig. 8). As for problems of quality, development or evolution of software (AQ4), the studies seek solutions through an artifact (tool, framework, methodology, etc.), and most of them were evaluated with '5' (see Fig. 8). In relation to the proposals being able to be implemented in industrial environments (AQ5), they were evaluated with a high value ('5'). This means that a study is considered replicable, but there are strong dependencies in terms of tools, software and configurations that should be considered (see Fig. 8).

Finally, questions AQ6 and AQ7 objectively assess the citations and relevance of the conferences and journals in which the selected studies were published (see Fig. 8). Most studies have been referenced several times. Table 11 shows the most cited documents. The most cited paper is the study 2 (Abdalkareem *et al.*, 2017) (143 times). That study focuses on providing an insight into the potential impact of reusing repository code in mobile applications, through an exploratory study where open source applications in a

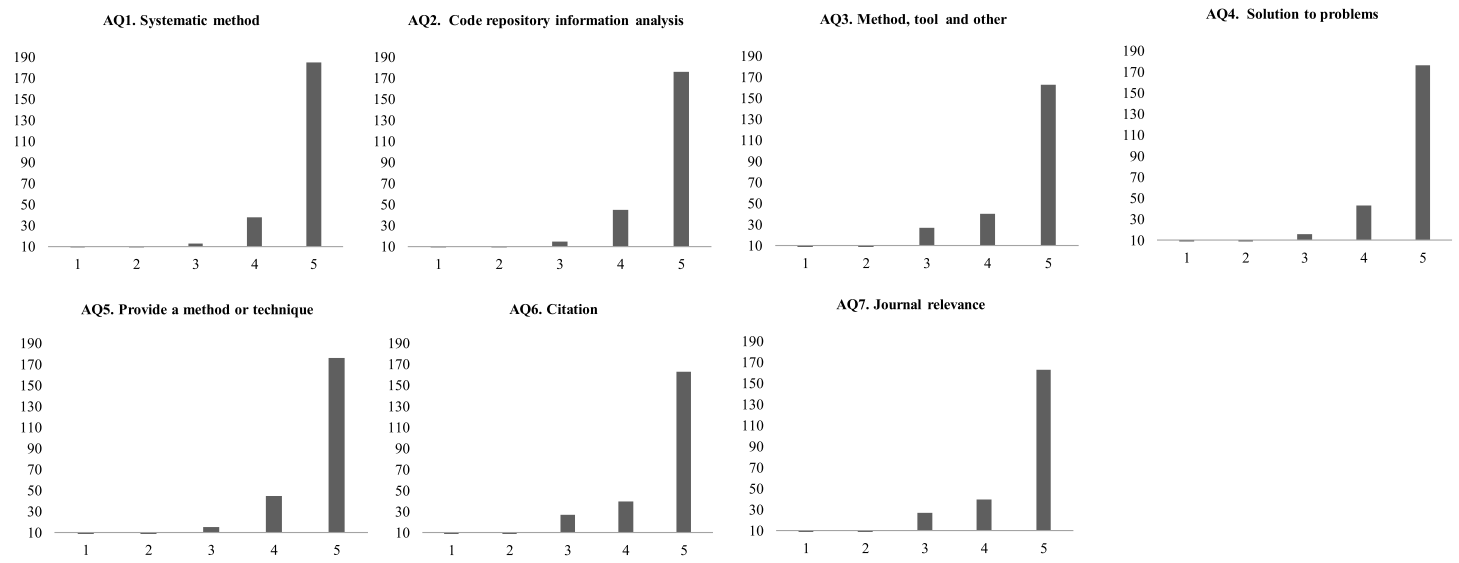

Fig. 8. Summary of the quality assessment.

Table 11
Most cited papers.

Study	# Citations	Year	AQ5
2	143	2017	5
186	72	2014	5
25	69	2014	5
9	66	2013	5
61	62	2012	5
187	46	2014	5
130	43	2012	5
165	42	2012	5
89	42	2014	5
159	40	2012	5

code repository are analysed. These results can benefit the research community in the development of new techniques and tools to facilitate and improve code reuse.

In addition to the analysis of the most cited articles, we have carried out a social network analysis (SNA), which allows us to generate a graph and identify the main groups of authors together with the most relevant authors in the area (Franco-Bedoya *et al.*, 2017). The methodology used for our analysis is based on (Wang *et al.*, 2016). The analysis is done with our SMS studies and is the main column of the network.

In the co-citation analysis, a matrix is compiled by retrieving the quotation counts of each pair of the important documents that were identified in the citation analysis, and a major component of the factor analysis is to reveal the knowledge clusters of the code repository research (Wang *et al.*, 2016).

We use the VOSviewer software that enables sophisticated cluster analysis without the need for in-depth knowledge of clusters and without the need for advanced computer skills (van Eck and Waltman, 2017).

In Fig. 9, the size of a cluster reflects the number of papers belonging to the cluster. Larger clusters include more publications. The distance between two clusters roughly indicates the relationship of the clusters in terms of citations. The clusters that are close to each other tend to be strongly related in terms of co-citations, while clusters that are farther apart tend to be less related (van Eck and Waltman, 2017). The curved lines between clusters also reflect the relationship between them, and the thickness of a line represents the number of citations between two clusters. VOSviewer has its own clustering technique (Waltman *et al.*, 2010). This clustering technique was used to divide in 14 clusters with four main branches. This was done based on the citation relationships between the analysed studies. In Fig. 9, each cluster has a colour indicating the group to which the cluster was assigned. Thus, a breakdown of papers concerning code repositories into broad subfields is obtained. A rough interpretation can be depicted as follows: The cluster in the down-left corner that becomes a branch (green, orange and pink nodes) seems to cover research about changes analysis, maintenance and code review to maintain software quality. The branch on the down-right (purple, red and brown nodes) seems to cover the research about code changes, commit analysis, automatic processing by focusing on bugs and software defects. The branch in top-left corner (blue and pink nodes) might be related

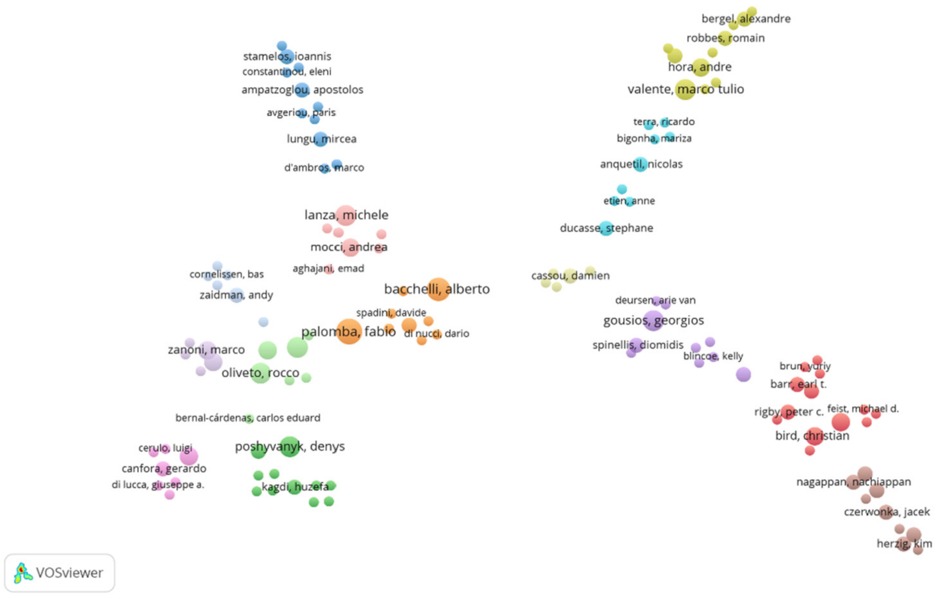

Fig. 9. Relationship between authors and research knowledge groups in code repositories.

to research in source code improvements and metrics/quality. Finally, the top-right branch (light blue and green) is more related to research in the field of defect/issue analysis, maintainability information and developer behaviour.

5. Discussion

This section presents the main results obtained through SMS for research and industry.

5.1. *Principal Findings*

The main research question and the reason for this SMS was to know the information that is extracted from the code repositories along with the methods and tools to process it and the output that is obtained from this process. Our SMS has scrutinized and schematized this field of research and determined its current status by analysing, evaluating, and understanding the research to date in relation to extraction, methods/tools and output generated from code repositories. The main findings are:

- **F1**. The research field regarding code repository analysis is in the process of improving its matureness. Researchers have worked in this discipline very hard in the last 10 years with several different proposals with some evidence. However, most of them have not been extensively applied in the industry. In addition, most of the papers have been published in high impact conferences and journals. Therefore, several objectives have been found to be covered by these research proposals. As a result, many of the proposals turn out to be innovative and built on previous research.

- **F2**. The authors consider several methods/techniques for the analysis of information obtained from code repositories. The collected studies point out that there are different techniques and methods from other research areas that can be applied in the analysis of information extracted from code repositories. We can detect some recurrent patterns. For example, the most recurrent techniques are those related to empirical or experimental analyses which are present for various inputs and outputs. Another insight is the extensive use of artificial intelligence and, more specifically, machine learning to analyse the information extracted from code repositories with good results. Finally, some tools and techniques combine some automatic processes with other sources to achieve the research goal.
- **F3**. The selected proposals contribute to the understanding of software quality and evolution. Several methods, techniques and tools have been found for the process of analysis of information extracted from code repositories, their application in the industry is given in a minimum measure and poor ascent, as it is demonstrated by some studies that give viability through empirical results. Although there are studies that go hand in hand between industry and academia, few initiatives are found in digital libraries.
- **F4**. The output obtained from the analysis of information from code repositories focuses on most studies to investigate the developer, such as classifying it or finding patterns of feelings (like through sentiment analysis) that infer from the coding. In summary, this analysis allows the developer to know the quality and evolution of the software beyond counting and measuring the lines of code and focuses on the human factor as a fundamental part within the software development.
- **F5**. Finally, another important output obtained from the analysis of information from code repositories is the analysis of changes. These studies focus on obtaining the impact of changes in the software code that is developed to try to find error patterns and to be able to make predictions of possible failures in the code before they occur. These changes in the code can greatly influence the quality and maintenance of the software, as well as have serious repercussions on costs and developers of the software project.

5.2. *Implications for Researchers and Practitioners*

The main findings presented above have some implications for researchers and practitioners working in the industry who research code repositories. For the academic world, the most used inputs for information analysis are mostly source code and commit data, the categories that have less amount of studies are an area of research to be explored. As for the methods and techniques used for the analysis of information in the repository, a wide variety of tools, techniques and methods are used (see Fig. 7), especially most approaches focus on empirical studies or artificial intelligence, which provides different approaches to information processing, and most studies seek to improve data processing to meet the studied objectives. As shown in Fig. 3, research involving analysis of information from code repositories is increasing every year. Therefore, it becomes a wide area of future research.

Another important implication for researchers is that most of the proposed methods and techniques require several tools and software to replicate the studies. This makes these techniques somewhat complex to replicate in the industry. Finally, researchers should also focus on how the obtained information is applied to solve problems of software quality and evolution, which is the important point for both academia and industry and therefore seeks to improve software development.

5.3. *Evaluation of Validity*

In this section we discuss the limitations of our systematic mapping study, based on the types of validity proposed by Petersen and Gencel (2013), which we describe below:

5.3.1. *Descriptive Validity*

"Descriptive validity refers to threats to the ability to capture and accurately represent the observations made" (Badampudi *et al.*, 2016). In an MSS, the main objective is to obtain available studies without any research bias. To avoid bias, we applied a review protocol, which was evaluated and approved by the authors as a means of quality assurance. As mentioned above, SMS guidelines of Kitchenham *et al.* (2011) and Petersen *et al.* (2015) were important for optimizing internal validity. The authors of this SMS double-checked the results of the selection procedure. These researchers took a random sample of 50% of the studies selected by the primary author and applied the inclusion/exclusion criteria for the selection procedure. To reduce the threats to data extraction, we created a form and used it in the pilot study (see Appendix B), which was validated by researchers. As well as Badampudi *et al.* (2016), the text of the selected primary studies was highlighted and marked, which made it easy to consult the document if a review was required. Finally, during the data extraction process, the researchers conducted several focus group sessions to discuss any potential controversies regarding quality assessment.

5.3.2. *Theoretical Validity*

Uncertainty of some factors by the author may affect theoretical validity (Badampudi *et al.*, 2016). In the case of our SMS, one of the main threats was the use of multiple terms and classifications to refer to code repositories. We mitigated this threat by using a synonym term defined in the search string, which was validated in the pilot search.

5.3.3. *Generalizability*

There are several limitations that can affect our SMS, this is the generalization presented by Petersen and Gencel (2013) and a distinction between internal and external generality. As far as systematic mapping is concerned, internal capacity is not a major threat (Petersen and Gencel, 2013), we believe that the most important limitation in our SMS is publication bias, because it is not possible to extract all the studies published in this area of research. We mitigate this threat by using five digital scientific databases considered relevant to software engineering recommended by Kuhrmann *et al.* (2017) as sources for study extraction. The consulted databases do not cover certain digital material that could be useful and relevant to our research, for example "technical reports, blog entries and video presentations" (Laukkanen *et al.*, 2017). Thus, this aspect is a strong limitation in research of code repositories in terms of research done by industry that is hardly ever published publicly. This limitation is reflected in low authorship between academia and industry. Anyway, this does not prove that the industry is not interested in analysing code repositories.

External generalizability measures the ability to generalize results, that is, the extent to which the results reported in a publication can be generalized in other contexts (Munir *et al.*, 2016; Wohlin *et al.*, 2012). In this regard, the main threat to external generalizability refers to our subjectivity in selecting, classifying, and understanding the point of view of the original authors of the studied works. A misperception or misunderstanding by us of a given paper may have led to a misclassification of the study. To minimize the chances of this, we apply a quality assurance system (Section 3.1.4). In addition, as we present the review protocol in detail in Section 3, our mapping is intended to be reliable to other investigators in terms of the search strategy, inclusion/exclusion criteria, and applied data extraction (Borg *et al.*, 2014).

5.4. *Interpretive Validity*

Interpretive validity is achieved when the drawn conclusions are reasonable from the data obtained and lead to the validation of the mapping (Petersen and Gencel, 2013). The main threat is author bias in the interpretation of the data. To mitigate this, the discussion groups and the classification process of the primary study selection were carried out. The researchers participated in various meetings to analyse and interpret the obtained data, in which their conclusions were discussed, and they made sure to maintain the same criteria.

5.5. *Reliability*

Repeatability is the ability of other researchers to replicate the results. To achieve reliability, research steps must be repeatable (Badampudi *et al.*, 2016). Detailed measures adopted in searches are limitations to theoretical validity and may lead to a lack of reporting capacity (Munir *et al.*, 2016). For example, the used search strings and databases extract the sought information, due to the documented inclusion/exclusion criteria, which increases reliability.

There is always a risk of losing primary studies with only one search string for all selected databases (Cosentino *et al.*, 2017). Therefore, a preliminary test with several versions of search strings was performed in the pilot search.

In addition, the inclusion/exclusion criteria were defined as in Genero *et al.* (2011). Collection of as many articles as possible aligned with the theme of the code repository. Repeatability of data extraction is important. We mitigated this threat by extracting and sorting the gathered papers in focus group sessions in which all researchers participated. The steps and information of our research are documented and published at https://GitHub.com/jaimepsayago/SMS including tables, graphs and the corresponding tables and analyses found in the document. In addition, all studies were classified according to the criteria of rigour and relevance adapted from Ivarsson and Gorschek (2011). This facilitates the traceability and repeatability of our study.

6. Conclusions

In this work, we conducted a SMS of the research published in the last eight years of five digital libraries. Through an extensive search and a systematic process, which has not been done in other similar studies, we have extracted and analysed data from more than 3700 papers, from which 236 documents have been selected. Relevant papers have been systematically analysed for answering the questions posed in this research.

This study reveals some trends in the current use of the evolving software coding and the massive use of code repositories as a platform for software development. These projects can range from an academic practice to large enterprise software projects. This allows us to analyse the information from these repositories, such as obtaining patterns, metrics and predictions in software development.

We believe that the conducted research is useful for developers working on software development projects that seek to improve maintenance and understand the evolution of software through the usage and analysis of the code repositories.

One important contribution is that we have defined a taxonomy that was divided according to input, method and output of the analysed proposals. Through this mapping study, we have identified the main information inputs used within code repositories that are commonly analysed: source code and commit information (RQ1).

A wide variety of tools and methods were used for the processing of information extracted from the code repository, especially most studies focus on using empirical and other experimental analyses, but also researchers are aligned to different approaches to information processing used in other fields of research such as artificial intelligence, with a special mention to machine learning. Together with these, data mining and other automatic techniques are employed to improve data processing in code repositories to meet the investigated objectives (RQ2).

Our analysis also raises the type of information derived from the processing of information from the code repository. In this sense, most studies are focused on investigating the developer behaviour or change analysis (RQ3). The analysis of the developer behaviour allows to know the quality and evolution of the software beyond counting and measuring the lines of code and focuses on the most important factor in software development. Meanwhile, the change analysis focuses on obtaining the impact of changes in the code of the software being developed, in order to try to find patterns of errors and to be able to make predictions of possible failures in the code.

In future work, we will focus on investigating the areas that have not yet been taken into consideration and that were identified in this systematic mapping study. We will attempt to directly research about artifacts for developer analysis. Finally, we will focus our research efforts on the analysis, measurement or testing of artifacts to determine and predict the impact on software quality from developer sentiments.

A. Search Strings

This appendix shows the search strings with specific syntax for the digital libraries used in the systematic mapping study (Table 12).

Table 12
Concrete syntax of the search string for each digital library.

Source	Search String
Scopus	TITLE-ABS-KEY ((“code repository” OR “software repository” OR “version control system” OR “GIT” OR “SVN”) AND (“analysis” OR “inspection” OR “mining” OR “exploring”)) AND (LIMIT-TO (DOCTYPE, “ar”) OR LIMIT-TO (DOCTYPE, “cp”) OR LIMIT-TO (DOCTYPE, “cr”) OR LIMIT-TO (DOCTYPE, “re”) OR LIMIT-TO (DOCTYPE, “ch”) OR LIMIT-TO (DOCTYPE, “bk”)) AND (LIMIT-TO (SUBJAREA, “COMP”) OR LIMIT-TO (SUBJAREA, “ENGI”) OR LIMIT-TO (SUBJAREA, “MATH”) OR LIMIT-TO (SUBJAREA, “DECI”)) AND (LIMIT-TO (PUBYEAR, 2020) OR LIMIT-TO (PUBYEAR, 2019) OR LIMIT-TO (PUBYEAR, 2018) OR LIMIT-TO (PUBYEAR, 2017) OR LIMIT-TO (PUBYEAR, 2016) OR LIMIT-TO (PUBYEAR, 2015) OR LIMIT-TO (PUBYEAR, 2014) OR LIMIT-TO (PUBYEAR, 2013) OR LIMIT-TO (PUBYEAR, 2012)) AND (LIMIT-TO (LANGUAGE, “English”))
IEEE Xplore	(((“Document Title”:”code repository” OR “software repository” OR “version control system” OR git OR svn AND analysis) AND “Abstract”:”code repository” OR “software repository” OR “version control system” OR git OR svn AND analysis) AND “Author Keywords”:”code repository” OR “software repository” OR “version control system” OR git OR svn AND analysis)
ACM Digital Library	“query”: {acmdlTitle:(code repository software repository git svn) AND acmdlTitle:(analysis inspection mining exploring) AND recordAbstract:(code repository software repository git svn) AND recordAbstract:(analysis inspection mining exploring) AND keywords.author.keyword:(code repository software repository git svn) AND keywords.author.keyword:(analysis inspection mining exploring) }”filter”: {”publicationYear”:{ “gte”:2012, “lte”:2019 }},{owners.owner = HOSTED}
Science Direct	(“code repository” OR “software repository” OR “version control system” OR git OR svn) AND (analysis OR inspection OR mining OR exploring)
ISI Web of Science	TS = (“code repository” OR “software repository” OR “version control system” OR git OR svn) AND TS = (analysis OR inspection OR mining OR exploring)

B. Data Extraction Form

Information	RQ/AQ
Meta-Information	
number	
Author	
Title	
Abstract	
Keywords	
Conference/Journal	
Year	
Reference Type	
DOI	
Tracking information about the selection of primary studies	
. . .	
Classification	
Type Information Extract	RQ1
Methods/Techniques	RQ2
Type Information Result	RQ3

(*continued on next page*)

Table 12
(*continued*)

Information	RQ/AQ
Type of Research	RQ4
Industry/Academia	RQ5
Journal and Conference Relevance/Citations	
ERA	
QUALIS	
JCR	
Q-JCR	
cited by (*Scopus)	
Score	
Quality Assessment	
Information Extract	AQ1
Information Result	AQ2
Methods/Techniques	AQ3
Solution Problem	AQ4
Application	AQ5
Q-cited	AQ6
Relevance of Conference/Journal	AQ7

C. Selected Primary Studies

It is possible to view the select primary studies obtained to SMS in the next link https://GitHub.com/jaimepsayago/SMS.

Funding

This study has been partially funded by the G3SOFT (SBPLY/17/180501/000150), GEMA (SBPLY/17/180501/000293) and SOS (SBPLY/17/180501/000364) projects funded by the 'Dirección General de Universidades, Investigación e Innovación – Consejería de Educación, Cultura y Deportes; Gobierno de Castilla-La Mancha'. This work is also a part of the projects BIZDEVOPS-Global (RTI2018-098309-B-C31) and ECLIPSE (RTI2018-094283-B-C31) funded by Ministerio de Economía, Industria y Competitividad (MINECO) & Fondo Europeo de Desarrollo Regional (FEDER).

References

Abdalkareem, R. Shihaba, E., Rillingb, J. (2017). On code reuse from StackOverflow: an exploratory study on Android apps. *Information and Software Technology*, 88, 148–158. https://doi.org/10.1016/j.infsof.2017.04.005.

Abdeen, H. Bali, K., Sahraoui, H., Dufour, B. (2015). Learning dependency-based change impact predictors using independent change histories. *Information and Software Technology*, 67, 220–235. https://doi.org/10.1016/j.infsof.2015.07.007.

Abuasad, A., Alsmadi, I.M. (1994, (2012)). The correlation between source code analysis change recommendations and software metrics. In: *ICICS '12: Proceedings of the 3rd International Conference on Information and Communication Systems*. https://doi.org/10.1145/2222444.2222446.

Agarwal, H., Husain, F., Saini, P. (2019). Next generation noise and affine invariant video watermarking scheme using Harris feature extraction. In: *Third International Conference, ICACDS 2019, Ghaziabad, India, April 12–13, 2019, Revised Selected Papers, Part II, Advances in Computing and Data Sciences*. Springer, Singapore, pp. 655–665. https://doi.org/10.1007/978-981-13-9942-8.

de Almeida Biolchini, J.C., Mian, P.G., Natali, A.C.C., Conte, T.U., Travassos, G.H. (2007). Scientific research ontology to support systematic review in software engineering. *Advanced Engineering Informatics*, 21(2), 133–151. https://doi.org/10.1016/j.aei.2006.11.006.

Amann, S., Beyer, S., Kevic, K., Gall, H. (2015). Software mining studies: goals, approaches, artifacts, and replicability. In: Meyer, B., Nordio, M. (Eds.), *Software Engineering. LASER 2013, LASER 2014*, Lecture Notes in Computer Science, Vol. 8987. Springer, Cham. https://doi.org/10.1007/978-3-319-28406-4_5.

Arora, R., Garg, A. (2018). Analysis of software repositories using process mining. *Smart Computing and Informatics Smart Innovation, Systems and Technologies*, 78, 637–643. https://doi.org/10.1007/978-981-10-5547-8_65.

Badampudi, D., Wohlin, C., Petersen, K. (2016). Software component decision-making: in-house, OSS, COTS or outsourcing – a systematic literature review. *Journal of Systems and Software*, 121, 105–124. https://doi.org/10.1016/j.jss.2016.07.027.

Bailey, J., Budgen, D., Turner, M., Kitchenham, B., Brereton, P., Linkman, S. (2007). Evidence relating to object-oriented software design: a survey. In: *Proceedings of the First International Symposium on Empirical Software Engineering and Measurement*. IEEE Computer Society, USA, pp. 482–484. https://doi.org/10.1109/ESEM.2007.46.

Ball, T., Kim J., Siy H.P., (1997). If your version control system could talk. In: *ICSE Workshop on Process Modelling and Empirical Studies of Software Engineering* https://doi.org/10.1.1.48.910.

Baltrusaitis, T. Ahuja, C., Morency, L. (2019). Multimodal machine learning: a survey and taxonomy. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 41(2), 423–443. https://doi.org/10.1109/TPAMI.2018.2798607.

Barredo Arrieta, A., Díaz-Rodríguez, N., Del Ser, J., Bennetot, A., Tabik, S., Barbado, A., Garcia, S., Gil-Lopez, S., Molina, D., Benjamins, R., Chatila, R., Herrera, F. (2020). Explainable Explainable Artificial Intelligence (XAI): concepts, taxonomies, opportunities and challenges toward responsible AI. *Information Fusion*, 58, 82–115. https://doi.org/10.1016/j.inffus.2019.12.012.

Borg, M., Runeson, P., Ardö A. (2014). Recovering from a decade: a systematic mapping of information retrieval approaches to software traceability. *Empirical Software Engineering*, 19(6), 1565–1616. https://doi.org/10.1007/s10664-013-9255-y.

Borges, H., Tulio Valente, M. (2018). What's in a GitHub Star? Understanding repository starring practices in a social coding platform. *Journal of Systems and Software*, 146, 112–129. https://doi.org/10.1016/j.jss.2018.09.016.

Cavalcanti, Y.C. da Mota Silveira Neto, P.A., do Carmo Machado, I., Vale, T.F., de Almeida, E.S., de Lemos Meira, S.R. (2014). Challenges and opportunities for software change request repositories: a systematic mapping study. *Journal of Software: Evolution and Process*, 26(7), 620–653. https://doi.org/10.1002/smr.1639.

Chahal, K.K., Saini, M. (2016). Open source software evolution: a systematic literature review (Part 1). *International Journal of Open Source Software and Processes*, 7(1), 1–27. https://doi.org/10.4018/IJOSSP.2016010101.

Chaturvedi, K.K., Sing, V.B., Singh, P. (2013). Tools in mining software repositories. In: *Proceedings of the 2013 13th International Conference on Computational Science and Its Applications, ICCSA 2013*, pp. 89–98. https://doi.org/10.1109/ICCSA.2013.22.

Chen, T.H. Thomas, S.W., Hassan, A.E. (2016). A survey on the use of topic models when mining software repositories. *Empirical Software Engineering*. https://doi.org/10.1007/s10664-015-9402-8.

Cornelissen, B. Zaidman, A., van Deursen, A., Moonen, L., Koschke, R. (2009). A systematic survey of program comprehension through dynamic analysis. *IEEE Transactions on Software Engineering*, 35(5), 684–702. https://doi.org/10.1109/TSE.2009.28.

Cosentino, V., Cánovas Izquierdo J.L. Cabot J. (2017). A systematic mapping study of software development with GitHub. *IEEE Access*, 5, 7173–7192. https://doi.org/10.1109/ACCESS.2017.2682323.

Costa, C., Murta, L. (2013). Version control in Distributed Software Development: a systematic mapping study. In: *IEEE 8th International Conference on Global Software Engineering, ICGSE 2013*, pp. 90–99. https://doi.org/10.1109/ICGSE.2013.19.

Datta, S., Datta, S., Naidu, K.V.M. (2012). Capacitated team formation problem on social networks. In: *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1005–1013. https://doi.org/10.1145/2339530.2339690.

De Farias, Novais, M.A.F.R., Colaço Júnior, M., da Silva Carvalho, L.P. (2016). A systematic mapping study on mining software repositories. In: *SAC '16: Proceedings of the 31st Annual ACM Symposium on Applied Computing*, pp. 1472–1479. https://doi.org/10.1145/2851613.2851786.

Del Carpio, P.M. (2017). Extracción de Nubes de Palabras en Repositorios Git. *2017 12th Iberian Conference on Information Systems and Technologies (CISTI)* https://doi.org/10.23919/CISTI.2017.7975911.

Demeyer, S. Murgia, A., Wyckmans, K., Lamkanfi, A. (2013). Happy birthday! A trend analysis on past MSR papers. In: *2013 10th Working Conference on Mining Software Repositories (MSR)*, pp. 353–362. https://doi.org/10.1109/MSR.2013.6624049.

Dias de Moura, M.H., Dantas do Nascimento H.A., Couto Rosa T. (2014). Extracting new metrics from version control system for the comparison of software developers. In: *ARES '14: Proceedings of the 2014 Ninth International Conference on Availability, Reliability and Security*, pp. 41–50. https://doi.org/10.1109/SBES.2014.25.

Dias, M., Bacchelli, A., Gousios, G., Cassou, D., Ducasse, S., (2015). Untangling fine-grained code changes. In: *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, pp. 341–350. https://doi.org/10.1109/SANER.2015.7081844.

Dit, B., Revelle, M., Gethers, M., Poshyvanyk, D. (2013). Feature location in source code: a taxonomy and survey. *Journal of Software: Evolution and Process*, 25(1), 53–95. https://doi.org/10.1002/smr.567.

Dyer, R., Nguyen, H.A., Rajan, H., Nguyen, T.N. (2015). Boa: Ultra-large-scale software repository and source-code mining. *ACM Transactions on Software Engineering and Methodology*, 25, 1. https://doi.org/10.1145/2803171.

Elsen, S. (2013). VisGi: visualizing Git branches. *2013 First IEEE Working Conference on Software Visualization (VISSOFT)*. https://doi.org/10.1109/VISSOFT.2013.6650522.

Falessi, D., Reichel, A. (2015). Towards an open-source tool for measuring and visualizing the interest of technical debt. In: *2015 IEEE 7th International Workshop on Managing Technical Debt (MTD)*, pp. 1–8. https://doi.org/10.1109/MTD.2015.7332618.

Farias, M., Novais, R., Ortins, P., Colaço, M., Mendonça, M. (2015). Analyzing distributions of emails and commits from OSS contributors through mining software repositories: an exploratory study. In: *ICEIS 2015: Proceedings of the 17th International Conference on Enterprise Information Systems,* Vol. *2*, pp. 303–310. https://doi.org/10.5220/0005368603030310.

Feldt, R., de Oliveira Neto, F.G., Torkar, R. (2018). Ways of applying artificial intelligence in software engineering. In: *2018 IEEE/ACM 6th International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering (RAISE)*, pp. 35–41.

Finlay, J. Pears, R., Connor, A.M. (2014). Data stream mining for predicting software build outcomes using source code metrics. *Information and Software Technology*, 56(2), 183–198. https://doi.org/10.1016/j.infsof.2013.09.001.

Foucault, M., Palyart, M., Blanc, X., Murphy, G.C., Falleri, J.-R. (2015). Impact of developer turnover on quality in open-source software. In: *ESEC/FSE 2015: Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, pp. 829–841. https://doi.org/10.1145/2786805.2786870.

Franco-Bedoya, O. Ameller, D., Costal, D., Franch, X. (2017). Open source software ecosystems: a systematic mapping. *Information and Software Technology*, 91, 160–185. https://doi.org/10.1016/j.infsof.2017.07.007.

Fu, Y., Yan, M., Zhang, X., Xu, L., Yang, D., Kymer, J.D. (2015). Automated classification of software change messages by semi-supervised Latent Dirichlet Allocation. *Information and Software Technology*, 57(1), 369–377. https://doi.org/10.1016/j.infsof.2014.05.017.

Gamalielsson, J., Lundell, B. (2014). Sustainability of Open Source software communities beyond a fork: How and why has the LibreOffice project evolved? *Journal of Systems and Software*, 89(1), 128–145. https://doi.org/10.1016/j.jss.2013.11.1077.

Gani, A. Siddiqa, A., Shamshirband, S., Hanum, F. (2016). A survey on indexing techniques for big data: taxonomy and performance evaluation. *Knowledge and Information Systems*, 46(2), 241–284. https://doi.org/10.1007/s10115-015-0830-y.

Genero, M., Fernandez, A.M., James Nelson, H., Poels, G. (2011). A systematic literature review on the quality of UML models. *Journal of Database Management*, 22(3), 46–66. https://doi.org/10.4018/jdm.2011070103.

Genero Bocco M.F., Cruz-Lemus J.A., Piattini Velthuis M.G. (2014). *Métodos de investigación en ingeniería del software*. Ra-Ma.

Grossi, V., Romei, A., Turini, F. (2017). Survey on using constraints in data mining. *Data Mining and Knowledge Discovery*, 31(2), 424–464. https://doi.org/10.1007/s10618-016-0480-z.

Güemes-Peña, D., López-Nozal, C., Marticorena-Sánchez, R. (2018). Emerging topics in mining software repositories: machine learning in software repositories and datasets. *Progress in Artificial Intelligence*, 7, 237–247. https://doi.org/10.1007/s13748-018-0147-7.

Gupta, M., Sureka, A., Padmanabhuni, S. (2014). Process mining multiple repositories for software defect resolution from control and organizational perspective. In: *MSR 2014: Proceedings of the 11th Working Conference on Mining Software Repositories*, pp. 122–131. https://doi.org/10.1145/2597073.2597081.

Haddaway, N.R., Macura, B., Whaley, P. (2018). ROSES Reporting standards for Systematic Evidence Syntheses: Pro forma, flow-diagram and descriptive summary of the plan and conduct of environmental systematic reviews and systematic maps. *Environmental Evidence*, 7(1), 4–11. https://doi.org/10.1186/s13750-018-0121-7.

Harman, M. (2012). The role of artificial intelligence in software engineering. In: *2012 First International Workshop on Realizing AI Synergies in Software Engineering (RAISE)*. IEEE. https://doi.org/10.1109/RAISE.2012.6227961.

Hassan, A.E. (2008). The road ahead for mining software repositories. In: *2008 Frontiers of Software Maintenance*, pp. 48–57. https://doi.org/10.1109/FOSM.2008.4659248. 2008.

Herzig, K., Just S., Zeller A., (2016). The impact of tangled code changes on defect prediction models. *Empirical Software Engineering* , 21(2), 303–336. https://doi.org/10.1007/s10664-015-9376-6.

Hidalgo Suarez, C.G., Bucheli, V.A., Restrepo-Calle, F., Gonzalez, F.A. (2018). A strategy based on technological maps for the identification of the state-of-the-art techniques in software development projects: Virtual judge projects as a case study. In: Serrano, C.J., Martínez-Santos, J. (Eds.), *Advances in Computing. CCC 2018*, Communications in Computer and Information Science, Vol. 885. Springer, Cham. https://doi.org/10.1007/978-3-319-98998-3_27.

Ivarsson, M., Gorschek, T. (2011). A method for evaluating rigor and industrial relevance of technology evaluations. *Empirical Software Engineering*, 16(3), 365–395. https://doi.org/10.1007/s10664-010-9146-4.

Jarczyk, O., Jaroszewicz, S., Wierzbicki, A., Pawlak, K., Jankowski-Lorek, M. (2017). Surgical teams on GitHub: modeling performance of GitHub project development processes. *Information and Software Technology*, 100, 32–46. https://doi.org/10.1016/j.infsof.2018.03.010. 2018.

Jiang, J., Lo, D., Zheng, J., Xia, X., Yang, Y., Zhang, L., (2019). Who should make decision on this pull request? Analyzing time-decaying relationships and file similarities for integrator prediction. *Journal of Systems and Software*, 154, 196–210. https://doi.org/10.1016/j.jss.2019.04.055.

Joblin, M., Apel, S., Riehle, D., Mauerer, W., Siegmund, J. (2015). From developer networks to verified communities: a fine-grained approach. In: *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering (ICSE)*, pp. 563–573. https://doi.org/10.1109/ICSE.2015.73.

Joy, A., Thangavelu, S., Jyotishi, A. (2018). Performance of GitHub open-source software project: an empirical analysis. In: *2018 Second International Conference on Advances in Electronics, Computers and Communications (ICAECC)*, pp. 1–6. https://doi.org/10.1109/ICAECC.2018.8479462.

Just, S., Herzig, K., Czerwonka, J., Murphy, B. (2016). Switching to git: the good, the bad, and the ugly. In: *2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE)*, pp. 400–411. https://doi.org/10.1109/ISSRE.2016.38.

Kagdi, H., Collard, M.L., Maletic, J.I. (2007). A Survey and Taxonomy of Approaches for Mining Software Repositories in the Context of Software Evolution. *Journal of Software: Evolution and Process*, 19(2), 77–131. https://doi.org/10.1002/smr.344.

Kagdi1, H., Gethers, M., Poshyvanyk, D., Hammad, M. (2014). Assigning change requests to software developers. *Journal of Software: Evolution and Process*, 26(12), 1172–1192. https://doi.org/10.1002/smr.530.

Kalliamvakou, E., Gousios, G., Blincoe, K., Singer, L., German, D.M., Damian, D. (2016). An in-depth study of the promises and perils of mining GitHub. *Empirical Software Engineering*, 21(5), 2035–2071. https://doi.org/10.1007/s10664-015-9393-5.

Kasurinen, J., Knutas, A. (2018). Publication trends in gamification: a systematic mapping study. *Computer Science Review*, 27, 33–44. https://doi.org/10.1016/j.cosrev.2017.10.003.

Kirinuki, H., Higo, Y., Hotta, K., Kusumoto, S. (2014). Hey! Are you committing tangled changes? In: *ICPC 2014: Proceedings of the 22nd International Conference on Program Comprehension*, pp. 262–265. https://doi.org/10.1145/2597008.2597798.

Kitchenham, B. (2007). *Guidelines for performing Systematic Literature Reviews in Software Engineering*. https://doi.org/10.1145/1134285.1134500.

Kitchenham, B., Pearl Brereton, O., Budgen, D., Turner, M., Bailey, J., Linkman, S. (2009). Systematic literature reviews in software engineering – a systematic literature review. *Information and Software Technology*, 51(1), 7–15. https://doi.org/10.1016/j.infsof.2008.09.009.

Kitchenham, B., Sjøberg, D.I.K., Dyba, T., Pearl Brereton, O., Budgen, D., Höst, M. (2013). Trends in the quality of human-centric software engineering experiments – a quasi-experiment. *IEEE Transactions on Software Engineering*, 39(7), 1002–1017. https://doi.org/10.1109/TSE.2012.76.

Kitchenham, B.A., Budgen, D., Pearl Brereton, O. (2011). Using mapping studies as the basis for further research – a participant-observer case study. *Information and Software Technology*, 53(6), 638–651. https://doi.org/10.1016/j.infsof.2010.12.011.

Kuhrmann, M., Méndez Fernández, D., Daneva, M. (2017). On the pragmatic design of literature studies in software engineering: an experience-based guideline. *Empirical Software Engineering*, 22(6), 2852–2891. https://doi.org/10.1007/s10664-016-9492-y.

Kumar, L., Sripada, S.K., Sureka, A., Rath, S.K. (2018). Effective fault prediction model developed using Least Square Support Vector Machine (LSSVM). *Journal of Systems and Software*, 137, 686–712. https://doi.org/10.1016/j.jss.2017.04.016.

Laukkanen, E., Itkonen, J., Lassenius, C. (2017). Problems, causes and solutions when adopting continuous delivery—a systematic literature review. *Information and Software Technology*, 82, 55–79. https://doi.org/10.1016/j.infsof.2016.10.001.

Lee, H., Seo, B., Seo, E. (2013). A git source repository analysis tool based on a novel branch-oriented approach. In: *2013 International Conference on Information Science and Applications (ICISA)*, pp. 1–4. https://doi.org/10.1109/ICISA.2013.6579457.

Li, H.Y., Li, M., Zhou, Z.-H. (2019). Towards one reusable model for various software defect mining tasks. *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, 11441 LNAI, 212–224. https://doi.org/10.1007/978-3-030-16142-2_17.

Liu, J., Li, J., He, L. (2016). A comparative study of the effects of pull request on GitHub projects. In: *2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC)*, pp. 313–322. https://doi.org/10.1109/COMPSAC.2016.27.

Malheiros, Y., Moraes, A., Trindade, C., Meira, S. (2012). A source code recommender system to support newcomers. In: *COMPSAC '12: Proceedings of the 2012 IEEE 36th Annual Computer Software and Applications Conference*, pp. 19–24. https://doi.org/10.1109/COMPSAC.2012.11.

Maqsood, J., Eshraghi, I., Sarmad Ali, S. (2017). Success or failure identification for GitHub's open source projects. In: *ICMSS '17: Proceedings of the 2017 International Conference on Management Engineering, Software Engineering and Service Sciences*, pp. 145–150. https://doi.org/10.1145/3034950.3034957.

Martínez-Torres, M.R., Toral, S.L., Barrero, F.J., Gregor, D. (2013). A text categorisation tool for open source communities based on semantic analysis. *Behaviour & Information Technology*, 32(6), 532–544. https://doi.org/10.1080/0144929X.2011.624634.

Munir, H., Wnuk K., Runeson P. (2016). Open innovation in software engineering: a systematic mapping study. *Empirical Software Engineering*, 21(2), 684–723. https://doi.org/10.1007/s10664-015-9380-x.

Murgia, A., Tourani, P., Adams, B., Ortu, M. (2014). Do developers feel emotions? An exploratory analysis of emotions in software artifacts. In: *MSR 2014: Proceedings of the 11th Working Conference on Mining Software Repositories*, pp. 262–271. https://doi.org/10.1145/2597073.2597086.

Negara, S., Codoban M., Dig D., Johnson R.E. (2014). Mining fine-grained code changes to detect unknown change patterns. In: *ICSE 2014: Proceedings of the 36th International Conference on Software Engineering*, pp. 803–813. https://doi.org/10.1145/2568225.2568317.

Nishi, M.A., Damevski, K. (2018). Scalable code clone detection and search based on adaptive prefix filtering. *Journal of Systems and Software*, 137, 130–142. https://doi.org/10.1016/j.jss.2017.11.039.

Novielli, N., Girardi D., Lanubile F. (2018). A benchmark study on sentiment analysis for software engineering research. In: *MSR '18: Proceedings of the 15th International Conference on Mining Software Repositories*, pp. 364–375. https://doi.org/10.1145/3196398.3196403.

Ozbas-Caglayan, K., Dogru, A.H. (2013). Software repository analysis for investigating design-code compliance. In: *2013 Joint Conference of the 23rd International Workshop on Software Measurement and the 8th International Conference on Software Process and Product Measurement*, pp. 231–233. https://doi.org/10.1109/IWSM-Mensura.2013.40.

Pedreira, O., García, F., Brisaboa, N., Piattini, M. (2015). Gamification in software engineering – a systematic mapping. *Information and Software Technology*, 57(1), 157–168. https://doi.org/10.1016/j.infsof.2014.08.007.

Perez-Castillo, R., Ruiz-Gonzalez, F., Genero, M., Piattini, M. (2019). A systematic mapping study on enterprise architecture mining A systematic mapping study on enterprise architecture mining. *Enterprise Information Systems*, 13(5), 675–718. https://doi.org/10.1080/17517575.2019.1590859.

Petersen, K., Gencel, C. (2013). Worldviews, research methods, and their relationship to validity in empirical software engineering research. In: *2013 Joint Conference of the 23rd International Workshop on Software Measurement and the 8th International Conference on Software Process and Product Measurement*, pp. 81–89. https://doi.org/10.1109/IWSM-Mensura.2013.22.

Petersen, K., Feldt, R., Mujtaba, S., Mattsson, M. (2008). Systematic mapping studies in software engineering. In: *EASE'08: Proceedings of the 12th international conference on Evaluation and Assessment in Software Engineering*, pp. 68–77.

Petersen, K., Vakkalanka S. Kuzniarz L. (2015). Guidelines for conducting systematic mapping studies in software engineering: an update. *Information and Software Technology*, 64, 1–18. https://doi.org/10.1016/j.infsof.2015.03.007.

Rosen, C., Grawi, B., Shihab, E. (2015). Commit guru: analytics and risk prediction of software commits. In: *ESEC/FSE 2015: Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, pp. 966–969. https://doi.org/10.1145/2786805.2803183. 2015.

Shamseer, L. Moher, D., Clarke, M., Ghersi, D., Liberati, A., Petticrew, M., Shekelle, P., Stewart, L.A. (2015). Preferred reporting items for systematic review and meta-analysis protocols (PRISMA-P) 2015: elaboration and explanation. *The BMJ*, 349, 1–25. https://doi.org/10.1136/bmj.g7647.

Siddiqui, T., Ahmad, A. (2018). Data mining tools and techniques for mining software repositories: a systematic review. *Advances in Intelligent Systems and Computing*, 654, 717–726. https://doi.org/10.1007/978-981-10-6620-7_70.

Stol, K.J., Ralph P., Fitzgerald B. (2016). Grounded theory in software engineering research: a critical review and guidelines In: *ICSE '16: Proceedings of the 38th International Conference on Software Engineering*, pp. 120–131. https://doi.org/10.1145/2884781.2884833.

Tahir, A., Tosi, D., Morasca, S. (2013). A systematic review on the functional testing of semantic web services. *Journal of Systems and Software*, 86(11), 2877–2889. https://doi.org/10.1016/j.jss.2013.06.064.

van Eck, N.J., Waltman, L. (2017). Citation-based clustering of publications using CitNetExplorer and VOSviewer. *Scientometrics*, 111(2), 1053–1070. https://doi.org/10.1007/s11192-017-2300-7.

Waltman, L., van Eck N.J. Noyons Ed.C.M. (2010). A unified approach to mapping and clustering of bibliometric networks. *Journal of Informetrics*, 4(4), 629–635. https://doi.org/10.1016/j.joi.2010.07.002.

Wang, N., Liang, H. Jia, Y. Ge, S. Xue, Y. Wang, Z. (2016). Cloud computing research in the IS discipline: a citation/co-citation analysis. *Decision Support Systems*, 86, 35–47. https://doi.org/10.1016/j.dss.2016.03.006.

Wijesiriwardana, C., Wimalaratne, P. (2018). Fostering real-time software analysis by leveraging heterogeneous and autonomous software repositories. *IEICE Transactions on Information and Systems E*, 101D(11), 2730–2743. https://doi.org/10.1587/transinf.2018EDP7094.

Wohlin, C., Runeson, P., Höt, M., Ohlsson, M.C., Regnell, B., Wesslén, A. (2012). *Experimentation in Software Engineering*. Springer Publishing Company, Incorporated.

Wu, Y., Kropczynski, J., Shih, P.C., Carroll, J.M. (2014). Exploring the ecosystem of software developers on GitHub and other platforms. In: *CSCW Companion '14: Proceedings of the companion publication of the 17th ACM conference on Computer Supported Cooperative Work & Social Computing*, pp. 265–268. https://doi.org/10.1145/2556420.2556483.

Zolkifli, N.N., Ngah, A., Deraman, A. (2018). Version control system: a review. *Procedia Computer Science*, 135, 408–415. https://doi.org/10.1016/j.procs.2018.08.191.

J. Sayago-Heredia is a PhD student at the University of Castilla-La Mancha (UCLM), Spain. His research interests include software engineering. He is a professor at the School of Systems and Computing of the Pontificia Universidad Católica del Ecuador, Sede Esmeraldas. Contact him at jaime.sayago@pucese.edu.ec.

R. Perez-Castillo is a researcher at the Information Technologies and Systems Institute, University of Castilla-La Mancha (UCLM), Spain. His research interests include architecture-driven modernization, model-driven development, business-process archaeology, and enterprise architecture. Perez-Castillo received a PhD in computer science from UCLM. Contact him at ricardo.pdelcastillo@uclm.es.

M. Piattini is the director of the Alarcos Research Group and a full professor at the University of Castilla-La Mancha, Spain. His research interests include software and data quality, information-systems audit and security, and IT governance. Piattini received a PhD in computer science from Madrid Technical University, Spain. Contact him at mario.piattini@uclm.es.